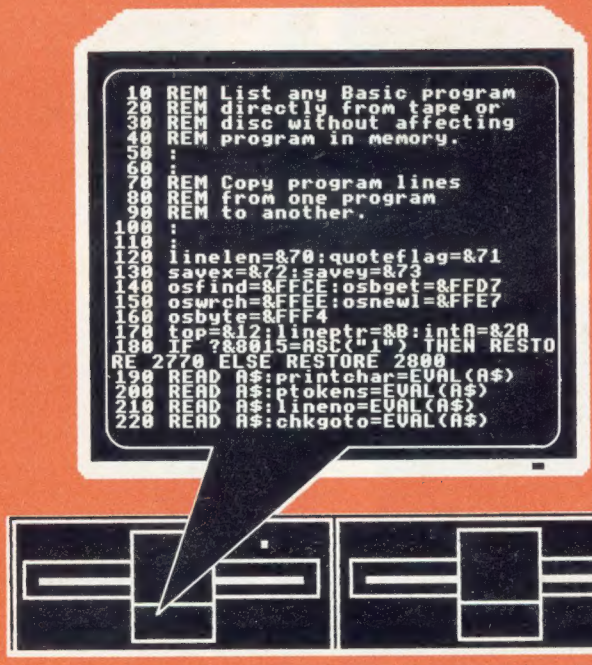
VOLUME 4 NUMBER 10 APRIL 1986 PRICE £1.30

# BEEBUG

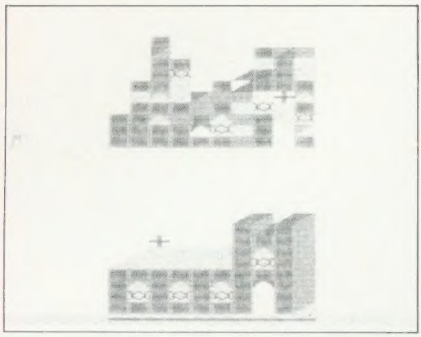
## FOR THE BBC MICRO

## DIRECT DISPLAY UTILITY

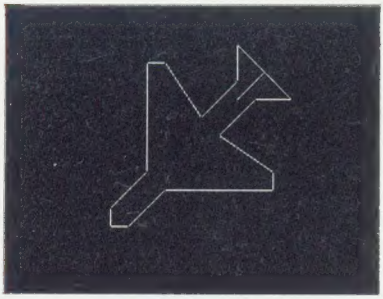


# BEEBUG

VOLUME 4 NUMBER 10
APRIL 1986

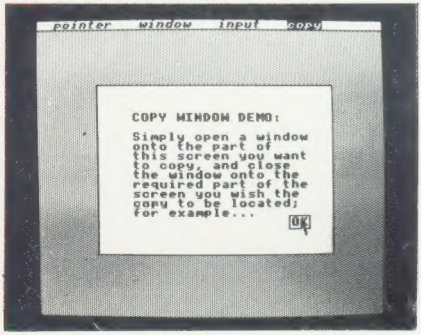



Jason Mason



Inbetweening

Windows and Icons




## GENERAL CONTENTS



## PROGRAMS



## HINTS AND TIPS

## FOUR YEARS ON

This issue of BEEBUG sees the end of our fourth year of publication. No other magazine for the BBC micro can boast such longevity nor the range of other features (discounts, free technical support, and so on) that BEEBUG offers you; and we intend to keep BEEBUG ahead of the rest in volume five. We're looking forward to the start of the new BEEBUG year with renewed enthusiasm, particularly now that the Master series offers new promise to Acorn aficionados.

In the coming months we shall be featuring a number of special articles and programs, both long and short, to keep you up to date and up to scratch with your BBC micro. These include:

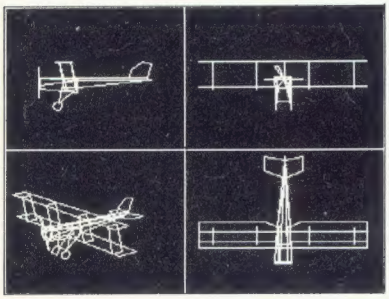
* Full feature sprite routines and an arcade game to use them
* BEEBUG Filer Follow-ups: Mailmerge & Data Converter for Wordwise Plus/View
* Short programs for special screen effects
* Airbrush drawing program
* Scrolling screen display
* Our biggest review yet of the printers you should consider
* Pop-up memo pad
* Sideways RAM utilities
* And many more.

So, if you want to stay ahead, stay with BEEBUG.

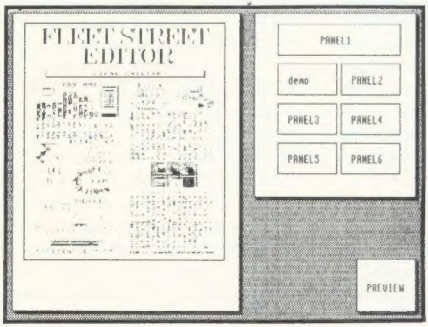
```
>*LINE 8,BASES
   10 REM Program BASES
   20 REM Version 1.0
   30 REM Author Ian Waugh
   40 REM BEEBUG April 1986
   50 REM Program Subject to Copyright
   60 :
  100 MODE7
  110 :
  120 PRINTCHR$(146);" DEC";TAB(8)"HEX";
TAB(17)"BINARY"
  130 VDU28,0,24,38,1
  140 :
  150 FOR num=1 TO 256
  160 dec$=STR$(num)
  170 PRINTTAB(5-LEN(dec$));dec$;
  180 hex$=FNbase(16,num)
>
>LIST
   10REM PROGRAM Basic List
   20REM VERSION 1.0
   30REM AUTHOR  Jagdish Sah
   40REM BEEBUG  APRIL 1986
   50REM PROGRAM SUBJECT TO COPYRIGHT
   60 :
  100PROCassemble
  110PROCchecksum
  120IF err% THEN END
>
```

Direct Display Utility

Interactive 3D



Fleet Street Editor




## EDITORIAL JOTTINGS

BEEBUG MAGAZINE CASSETTE/DISC

Every month, all the programs from BEEBUG magazine are collected together on disc and cassette. Often the versions recorded are expanded on the versions printed in the magazine and always they save an enormous amount of frequently frustrating time entering programs manually. Each month we also include the Magscan update for that month, and often one or more extras — particularly high quality machine code games that are unsuitable for publication in the magazine itself.

At only £3.00 for the cassette, or £4.75 for the disc (plus 50p P&P) this represents excellent value for money and to ensure prompt delivery, a 6 or 12 month subscription is recommended. Despite the recently announced price rise in the BEEBUG subscription we have managed to keep prices for the magazine cassette/disc at the same level as before — a small price to pay for so many useful, quality programs.

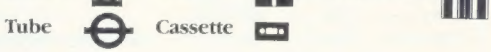
Full details of the contents of this month's magazine cassette/disc and full order details are given on the inside back cover.

## DISCOUNTS

We are able to add two of the products reviewed this month to our ever-increasing discount list for members. Acornsoft's excellent Micro Prolog is available for £75 and Mirrorsoft's Fleet Street Editor for £35. Both prices are inclusive of VAT and post and packing. A further £1 discount is available to personal callers to the BEEBUG shop.

## PROGRAM CLASSIFICATION

All programs in the magazine, and on the magazine cassette/disc, are marked with the symbols shown below. An uncrossed symbol indicates full working, a single line through symbol shows partial working for that configuration (normally some modifications will be required), and a cross through a symbol indicates a program that will not work on that type of system. There is a symbol for the B+ which includes the 128K version, and a symbol for the new Master series.



# News News News News News News Ne

### A Weather Eye

If you don't fancy investing in your own weather satellite receiving equipment (see last month's BEEBUG) then a subscription to Microlink will enable you to use someone else's. The latest addition to the Microlink service is Weatherlink. This enables you to receive weather pictures from the NOAA9 satellite via a central receiving station in Kent. Only a modem, downloading software, and a Microlink subscription are required to display weather satellite pictures on the screen of your Beeb. More details from Microlink on 061-429 8451.

### Big Ed

Full screen editors are in abundance these days. Another of this ilk comes in the form of Big-Ed. Modelled on the editor used in IBM mainframes, Big-Ed comes on ROM for £20. Features include split screen capability, search and replace, partial renumbering, and a real time clock. Further details from Lee Computers on 021-777 9631.

### I'm Alright GAC

If you've always fancied writing graphic adventures but never really had the know-how then Incentive Software's latest release is just up your street. GAC (Graphic Adventure Creator) is a utility to enable you to write full feature adventure games with your own graphics screens and features its own graphics designer and text compression routine. GAC costs £27.95 on disc and £22.95 on cassette from Incentive on 0734-591678.

### Fruit of the Peartree

The trouble with using your BBC micro for a small business is that the software soon costs more than the micro. Peartree software hope to solve that particular problem with an integrated suite of business software to handle all your needs for £99. The suite includes programs for stock control, customer database, purchase ledger, supplies database, sales ledger, nominal ledger, invoicing, quotations, bank account analysis, and petty cash control. If this sounds the business for your business, give Peartree a ring on 0480-50595.

### Paperware

There seems to be no end to the stream of 'create a newspaper with your Beeb' software. Ibbotsons Design Software has come up with 'Imagina' (Images in documents) along similar lines to Fleet Street Editor and Pagemaker but for the B+128K and Master. Imagina is also not limited to pages but can operate on a whole document of any length. Imagina costs £60 from Ibbotsons on 077 389-658.

### Another Master Series

Determined to master the universe, BBC Soft has released another book - 'Mastering Assembly Code' for £8.95. This latest addition to the 'Mastering' series takes you through the techniques involved in assembler programming and includes many routines for use in your own programs covering subjects such as graphics, sprites, events and interrupts, piracy protection and using the keyboard. BBC Soft is on 01-927 4518.

### More Logo

LSL claims to bring Logo 'out of the ghetto of turtle graphics and sprites' with more applications software to run with the LSL Logo ROM (see review in BEEBUG Vol.3 No.10). Subjects covered include statistics, graph plotting, matrices, transformational geometry, sets, property lists, and data file handling. LSL has also developed a 'Technical LOGO' for the control of hardware interfaced to the Beeb. Further details on all this from LSL on 01-891 0989.

### A La Carte

A little like BEEBUG's Extended disc catalogue (Vol.4 No.s 2 and 5) is Microtest's MenuROM. For £12.95 this ROM claims to enable you to use disc software without knowing anything about computers. Menus are automatically created from any disc and programs selected run with a single key-press. Also provided are facilities to format, verify discs, and to repair corrupted ones. Further details from Microtest on 0208-3171.

# BEEBUG SOFT FORUM

## Studio 8 with Symphony

Studio 8, the new 8 track sound synthesizer program from BEEBUGSOFT has been upgraded. It is now available in a version which is compatible with the Symphony Keyboard from ATPL.

Existing users wishing to upgrade should return their disc/tape to the St. Albans office with payment of £4.00/ 2.00 respectively. The Symphony Keyboard is also available on special offer through BEEBUG Retail. Please call for details.

## Freelance Programmers

If you have any programs that you feel would complement the BEEBUGSOFT range, we would be pleased to hear from you now. Please write to The Software Manager at the St. Albans address. All submissions will be treated in strictest confidence.

## Demo Disc

To find out further information about the BEEBUGSOFT product range we strongly recommend our new Demo Disc. This disc gives us a unique chance to demonstrate exactly what our programs look and sound like when in use on your computer. The cost of £2.50 is more than covered by the money back voucher given with the disc. See the back cover of BEEBUG for further information.

## Magscan Updates

Each month on the magazine disc and cassette, we publish the Magscan data for the articles and reviews in that month's issue of BEEBUG. You may include this file in your Magscan program to keep it completely up to date. This is very easy to do using Wordwise or View. Wordwise users would proceed as follows:

1. Use Wordwise option 4 to load in volume 4 of Magscan.

2. Move to the end of the file and delete the last character. This will be a [ (i.e. the key to the right of @). Leave the cursor in this position.

3. Now Escape to the Wordwise menu and use option 4 to load in the latest Magscan file (filename similar to MSCN409) to the cursor position.

4. Move to the end of text and replace end of file marker (i.e. the [) as before.

5. Use option 1 to re-save the Magscan file.

## Dynamic Disc Prices Down

We are pleased to announce that after further negotiations with our disc suppliers we are able to pass on a significant reduction in price of the BEEBUG blank discs. We can fully recommend these discs, which are of the highest quality and now represent even greater value for money. Please check the Dynamic Disc advert for full details on the new prices.

## Romit Fast & Efficient Menus

Because short Romit-generated files stored on Eprom load with great speed, it is possible to create detailed and ornate mode 7 menus in a novel way.

Essentially the technique is to create your menu screen using a mode 7 screen editor such as the Teletext Pack, which will generate a screen dump to disc of your proposed menu screen. The first advantage of creating your menu screen in this way is the great flexibility that a mode 7 screen editor gives you in positioning your text, selecting columns, drawing borders and adding extra large characters etc.

Once the screen has been saved to disc, transfer it to RaFS Ram using the ROMIT *FILE command.

You can now write your menu program. This should start by selecting mode 7, and loading in the screen that you have just created. A simple GET routine can then be added to detect the appropriate keypress and respond as required, e.g..

```
 10 MODE7:*OPT1
 20 *L.SCREEN 7C00
 30 PRINTTAB(20,22)"choice:";
 40 REPEAT A$=GET$
 50 A%=INSTR("ABCDEF",A$)
 60 SOUND1,-15,20,2
 70 UNTIL A%
 80 PRINTA$
 90 ON A% GOTO 100,200,300,40
0,500
100 REM 1st menu choice
200 REM 2nd menu choice
300 REM 3rd menu choice
400 REM 4th menu choice
500 REM 5th menu choice
```

# POSTBAG

POSTBAG

### Dear Diary

BEEBUG members with single 40-track disc drives may like to know that it IS possible to get the whole of Chris Wragg's Personal Diary Manager program (BEEBUG Vol.4 No.7) onto one side of a 40-track disc despite the fact that when the file set-up program is run it generates a 'disc full' message. This is because a new file, irrespective of its final length, can only be opened if there is at least 16K free on the disc, and the file X.DIARY takes up 90K.

The solution is based on the fact that the first time a file is closed it releases any unused sectors. Thus if the set-up program sequence is reversed, so that the much shorter X.INDEX file is created first, both files can be accommodated, and will still leave just enough room on a 100K disc for the main program plus a one-line !BOOT file.

Alan Jones

### Meeting the Challenge

BEEBUG members who have recently purchased a Challenger 3 from Opus will probably have encountered difficulties if the EPROM is version 1.0. Opus have confirmed that a bug does exist and that they will exchange the EPROM for a new 1.01 version if the original is sent back to them first, promising a return-of-post service. It makes you wonder about the reviews published in (other) magazines which claim to have put this unit through its paces yet failed to notice the bug.

M.J.Keene

### Make that Call

James Fletcher's article on electronic mail in BEEBUG Vol.4 No.7 failed to mention that you can join Prestel without the microcomputing section for £6.50 a quarter and so have very cheap access to electronic mail. In off-peak times I can send 7 or 8 letters for the cost of one telephone unit.

Charles Lloyd

Very true, but we feel sure that many BEEBUG members enjoy much that Prestel Microcomputing has to offer and don't really begrudge the extra charges.

### Advanced Disc Toolkit

It seems that the reviewer of this ROM from ACP did not have an ADFS Beeb to hand when conducting his review (BEEBUG Vol.4 No.7). It is only then that the power of the chip becomes apparent.

The strength of this ROM is that the commands respond correctly regardless of what filing system you are using. Further, they successfully provide the utilities that Acorn put on disc for the ADFS. I seem to have lost my ADFS Utilities disc and have not noticed it has gone.

Its real power is the way it gives the ability to move files between filing systems, unlocking as you go, including tape. It is uncanny to be able to move from ADFS to DFS to TAPE, and move files between the three; most of this is the result of good design by Acorn but part is the ADT ROM itself.

Tim Powys-Lybbe

The ADT ROM was one of several to which we gave short reviews in the same issue. In retrospect we did probably give it much less attention than it deserves, although its value when used with an ADFS was more fully reported by Peter Rochford in his review of the ADFS (BEEBUG Vol.4 No.8). This is certainly a ROM worth looking at if you have both DFS and ADFS.

### Print Using Print Using

I was most interested in the articles by J.Pike in BEEBUG Vol.4 Nos.5 & 6 as the explanation of @% in the User Guide leaves much to be desired. The function FNUSING in the November issue appears to contain an error as in some cases a number can be printed with the wrong number of decimal places.

The problem can be overcome by replacing line 9090 in the routine with:

```
9090 format$=STR$(number)
```

This allows the value of @% set in line 9060 to control the conversion of number from "real" to "string".

P.J.S.Pauwels

Mr Pauwels is quite correct in spotting the loop hole in the FNUSING function and has highlighted both the problem and its solution.

# HINTS HINTS HINTS HINTS HINTS HINTS HINTS

### View, Wordwise Plus and Function Keys

From the main menu of Wordwise Plus or the command mode of View, you need not use Ctrl-Shift with a function key to access your definitions, just the function key alone should be pressed.

C.B. Hill

### Keyboard Input in an Event

As interrupts are disabled when an event servicing routine is being executed, the OSRDCH routine will not work. The solution for reading the keyboard during an event is to use OSBYTE &81 with Y=&FF and the negative INKEY number of the key to be detected in X.

Tim Powys-Lybbe

### ADE with Wordwise Plus

The on-screen underline and double strike features of Wordwise Plus will not operate if System's ADE ROM is in a higher priority to the Wordwise Plus ROM.

R.T. Bunnett

### Single Key Merge

This function key definition will merge a specified Basic program onto the end of one in memory and then renumber the complete program ready for listing or running.

```
*K.0I."Merge",P$:P$="*L
."+P$+" "+STR$~(TOP-2)+CHR$
13+"REN."+CHR$13:A%=138:X%=
0:F.C=0TOLENP$:Y%=ASC(M.P$,
C)):CA.&FFF4:N.|M
```

Bruce Roberts

### Those Wide Open Spaces

The following program will work (as expected) with or without the space after the zero in line 20.

```
10 N=0
20 IF N=0 D=1 ELSE D=2
30 PRINT D
```

However if the D's are changed to E's like this:

```
10 N=0
20 IF N=0 E=1 ELSE E=2
30 PRINT E
```

then the program must contain the space after the zero or a 'No FN at line 20' error is given.

The reason is very simple. Without the space the 'E' and '0' are treated together as signifying a number in exponential form so the '=' following it is taken as the last line of a function definition.

C. Procter

### Shadow Compatibility

If you wish to write disc software, compatible with the B, the B+ and the Master, a *SHADOW command will be needed in the !BOOT file to enable or disable the shadow RAM, but this will generate a 'Bad command' error on the model B. This can be avoided by creating a machine code program called SHADOW consisting only of an RTS instruction on the disc with:

```
?&70=&60:*SAVE SHADOW 70+1
```

C.W. Robertson

### Error Listing

To get a program to automatically list the line causing an error, use:

```
    10 ON ERROR GOTO 100
     .
     .
     .
   99 END
  100 ON ERROR OFF:MODE 7
  110 A$="K.0L."+STR$(ERL
)+"|M":$&900=A$:X%=0:Y%=&9:
CALL &FFF7
  120 *FX138,0,128
  130 END
```

Eric Pope

### More and More Modes

The Advanced User Guide contains a program (page 383) to produce a new graphics mode, dubbed mode 8, and this was reproduced in BEEBUG Vol.2 No.6. Here are the details for four more modes. All the possible graphics modes have been covered. These are text only modes like 3 and 6. Enter the relevant values from the table below into lines 60 to 140 or use them in your own routine.

| Mode | colms | clrs | RAM | ?&30A | ?&362 | ?&363 | ?&34F | ?&361 | ?&360 | basic mode | *FX154, |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 40 | 4 | 16K | 39 | &88 | &11 | &10 | 3 | 3 | 3 | 216 |
| 10 | 20 | 16 | 16K | 19 | &AA | &55 | &20 | 1 | &F | 3 | 244 |
| 11 | 20 | 4 | 8K | 19 | &88 | &11 | &10 | 3 | 3 | 6 | 196 |
| 12 | 10 | 16 | 8K | 9 | &AA | &55 | &20 | 1 | &F | 6 | 224 |

# Inbetweening

**How do you turn an aeroplane into a Christmas tree? Impossible? Take a look at O.R. Thomas's marvellously short program and learn all about inbetweening.**

Inbetweening is an amusing graphics display program that moulds one shape into another. Any shape is transformed into any other before your eyes, and then moulded into another, and another...

Just type in the program, run it, and sit back and watch the shapes contort themselves on your screen in smooth animation. This program gives an idea of the methods used nowadays in the cartoon business. The tedious job of drawing all the frames between a cartoon character's successive poses is today usually done by computer. You can experiment a little with the program and with your own shapes to produce complex opening sequences for your software (we have used a variation of this program in our own BEEBUG demonstration disc). Alternatively you can use it just to produce a very entertaining display.

The nine shapes already defined in the program can be easily added to within the limits of the Beeb's memory by adding more data. Additionally, the order in which the shapes are displayed and moulded one into another can also be readily changed.

THE TRANSFORMATION SEQUENCE

Each shape is transformed into another according to the order of the shape numbers in the DATA statement in line 1840 (the 'sequence' data). The shape numbers in this DATA statement are made up from the first number from each shape's data (in lines 1860 to 1950). The sequence data is terminated with either 0 or -1. The former denotes that the sequence is to be performed once only and the latter that it is to be repeated, the last shape in the sequence transforming into the first.

SHAPE DATA

All the data for the shapes is contained in DATA statements after the sequence data. Each shape may consist of up to twenty points and must be one that can be drawn as a continuous line, though this may join up at the ends. The first item in the data is the shape number (any number) used in the sequence data. The shapes can be defined in any order. The second item is the number of points in the shape and the third should be either 1 or -1 to determine the order in which a shape's points are drawn onto the screen. This affects the way in which that shape is transformed into the next (how much it is turned inside out) and can be altered to suit your own tastes and the shape concerned. You can try altering this value in the data given here to see the effect.

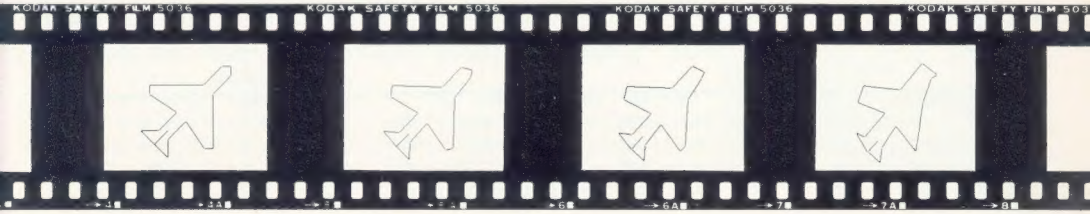
The data items following these first three are the x and y graphics co-ordinates of the shape points. So, for example, line 1900 defines shape 5 as a rectangle (five points) to be drawn in the order given in the DATA statement.

COMPETITION

If you fancy yourself as a high tech Walt Disney then enter our Inbetweening Competition. See page 23 for the details.

PROGRAM NOTES

The inbetween positions of the two shapes being moulded (called 'frames') are found by matching up points in the original shape with points in the final shape (if the number of points in each is unequal extra points are created to even up). The differences between each point's initial and final x and y co-ordinates are calculated and divided by 19 (the number of intervening frames). These are then



added to the initial set of co-ordinates to give the co-ordinates for each frame.

The different frames are simply animated using the well established method of using VDU19 to hide each frame as it is drawn or erased, and switch instantly between successive frames.

If you change the line numbering of the program, take care to alter the RESTORE statements at lines 1010 and 1050 to refer to the sequence data and those in 1060 and 1140 to refer to the first of the shape data.

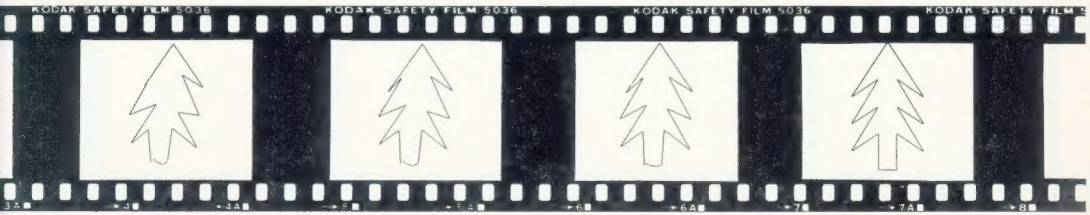
| | |
|---|---|
| PROCvariables | initializes variables. |
| PROCloadinfo | loads the required co-ordinates from DATA statements into the arrays. |
| PROCadd | creates new points to even up the numbers. |
| PROCworkout | calculates the frame co-ordinates. |
| PROCtrnsform | Organizes the smooth animation. |
| PROCdraw | draws a single frame. |

NOTE FOR DISC USERS

The program uses quite large arrays and therefore a lot of memory. Disc users will find that there is insufficient room for the program so they should set PAGE to &1200 (type PAGE=&1200 <Return>) before loading the program, or use a move down routine such as that accompanying the Burger Time game in last month's BEEBUG.

```
  10 REM PROGRAM INBETWEENING
  20 REM VERSION B2.0
  30 REM AUTHOR  O.R.THOMAS
  40 REM BEEBUG  APRIL 1986
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 ON ERROR GOTO220
 110 MODE1:VDU19,0,4;0;19,1,7;0;19,2,7;
0;19,3,7;0;23,1,0;0;0;0;
 120 DIM posx%(19,19),posy%(19,19),endx
%(19),endy%(19)
 130 PRINTTAB(14,1)"INBETWEENING"
 140 PROCvariables
 150 REPEAT
 160 PROCloadinfo:IF finish% GOTO190
 170 PROCworkout:PROCtransform
 180 seq%=seq%+1
 190 UNTIL finish%
 200 END
 210 :
 220 ON ERROR OFF
 230 MODE7:IF ERR<>17 REPORT:PRINT" at
line ";ERL
 240 END
 250 :
1000 DEF PROCloadinfo
1010 RESTORE1840
1020 FOR I%=1 TO seq%:READfrom%:NEXT
1030 READto%
1040 IF to%=0 finish%=TRUE:ENDPROC
1050 IF to%=-1 seq%=0:RESTORE1840:READt
o%
1060 RESTORE1860
1070 REPEAT
1080 READshapeno%,len1%,cont1%
1090 len1%=len1%-1:realen1%=len1%
1100 FOR I%=(-(cont1%=-1)*len1%) TO (-(
cont1%=1)*len1%) STEP cont1%
1110 READposx%(I%,0),posy%(I%,0)
1120 NEXT
1130 UNTIL shapeno%=from%
1140 RESTORE1860
1150 REPEAT
1160 READshapeno%,len2%,cont2%
1170 len2%=len2%-1:realen2%=len2%
1180 FOR I%=(-(cont2%=-1)*len2%) TO (-(
cont2%=1)*len2%) STEP cont2%
1190 READendx%(I%),endy%(I%)
1200 posx%(I%,19)=endx%(I%)
1210 posy%(I%,19)=endy%(I%)
1220 NEXT
1230 UNTIL shapeno%=to%
1240 IF len1%<>len2% PROCadd
1250 IF start% start%=FALSE:PROCdraw(0,
1)
1260 ENDPROC
1270 :
1280 DEF PROCadd
```

```
 1290 IF len1%<len2% posx%(len2%,0)=posx
%(len1%,0):posy%(len2%,0)=posy%(len1%,0)
 1300 IF len2%<len1% endx%(len1%)=endx%(
len2%):endy%(len1%)=endy%(len2%):GOTO137
0
 1310 FOR I%=len1% TO len2%-1
 1320 posx%(I%,0)=(posx%(len2%,0)-posx%(
len1%-1,0))/(len2%-len1%+1)*(I%-len1%+1)
+posx%(len1%-1,0)
 1330 posy%(I%,0)=(posy%(len2%,0)-posy%(
len1%-1,0))/(len2%-len1%+1)*(I%-len1%+1)
+posy%(len1%-1,0)
 1340 NEXT
 1350 len1%=len2%
 1360 ENDPROC
 1370 FOR I%=len2% TO len1%-1
 1380 endx%(I%)=(endx%(len1%)-endx%(len2
%-1))/(len1%-len2%+1)*(I%-len2%+1)+endx%
(len2%-1)
 1390 endy%(I%)=(endy%(len1%)-endy%(len2
%-1))/(len1%-len2%+1)*(I%-len2%+1)+endy%
(len2%-1)
 1400 NEXT
 1410 len2%=len1%
 1420 ENDPROC
 1430 :
 1440 DEF PROCworkout
 1450 FOR I%=0 TO 19
 1460 alterx=(endx%(I%)-posx%(I%,0))/19
 1470 altery=(endy%(I%)-posy%(I%,0))/19
 1480 FOR frame%=1 TO 18
 1490 posx%(I%,frame%)=posx%(I%,0)+(alte
rx*frame%)
 1500 posy%(I%,frame%)=posy%(I%,0)+(alte
ry*frame%)
 1510 NEXT
 1520 NEXT
 1530 posx%(realen1%,0)=posx%(len1%,0)
 1540 posy%(realen1%,0)=posy%(len1%,0)
 1550 ENDPROC
 1560 :
 1570 DEF PROCtransform
 1580 FOR frame%=1 TO 19
 1590 col%=col% EOR 3
 1600 VDU19,col%,4;0;
 1610 PROCdraw(frame%,col%)
 1620 VDU19,col%,7;0;
 1630 VDU19,(col% EOR 3),4;0;
 1640 PROCdraw(frame%-1,col% EOR 3)
 1650 NEXT
 1660 ENDPROC
 1670 :
 1680 DEF PROCdraw(pic%,col%)
 1690 GCOL3,col%
 1700 MOVEposx%(0,pic%),posy%(0,pic%)
 1710 any2%=len1%
 1720 IF pic%=0 any2%=realen1%
 1730 IF pic%=19 any2%=realen2%
 1740 FOR I%=1 TO any2%
 1750 DRAWposx%(I%,pic%),posy%(I%,pic%)
 1760 NEXT
 1770 ENDPROC
 1780 :
 1790 DEF PROCvariables
 1800 seq%=1:col%=1
 1810 start%=TRUE:finish%=FALSE
 1820 ENDPROC
 1830 REM----sequence---
 1840 DATA 8,6,7,8,9,10,4,5,2,1,3,4,-1
 1850 REM---shape data---
 1860 DATA 1,20,1, 1088,668,1088,604,768
,604,768,28,704,28,640,348,576,28,512,28
,512,604,192,604,192,668,576,668,576,732
,512,732,512,860,768,860,768,732,704,732
,704,668,1088,668:REM shape 1
 1870 DATA 2,16,1, 640,1024,896,640,704,
768,896,448,704,576,896,256,704,384,704,
128,576,128,576,384,384,256,576,576,384,
448,576,768,384,640,640,1024:REM shape 2
 1880 DATA 3,18,1, 512,334,416,238,512,1
42,512,270,640,398,768,206,832,206,832,5
90,960,718,960,782,896,782,768,654,384,6
54,384,590,576,462,448,334,320,334,416,2
38:REM shape 3
 1890 DATA 4,2,1, 640,900,640,100:REM sh
ape 4
 1900 DATA 5,5,1, 300,700,1000,700,1000,
200,300,200,300,700:REM shape 5
 1910 DATA 6,8,-1, 750,700,550,700,550,5
00,750,500,750,496,550,496,550,300,750,3
00:REM shape 6
 1920 DATA 7,9,-1, 750,700,550,700,550,5
00,750,500,750,496,550,496,550,300,754,3
00,754,300:REM shape 7
 1930 DATA 8,11,1, 550,500,710,500,750,5
40,750,660,710,700,550,700,550,300,710,3
00,750,340,750,460,710,500:REM shape 8
 1940 DATA 9,6,1, 550,700,550,340,590,30
0,710,300,750,340,750,700:REM shape 9
 1950 DATA 10,10,1, 750,660,710,700,590,
700,550,660,550,340,590,300,710,300,750,
340,750,500,650,500:REM shape 10
```



# DO-IT-YOURSELF BASIC

**Are you tired of Basic? Do you need functions and features which just aren't there? Alan Webster rejuvenates your Basic by adding all those extra features.**

This is the first part of a short series explaining how to extend BBC Basic with your own choice of keywords and commands. This month I shall introduce the method involved and provide a working example of a new command.

The main reasons for adding new commands to BBC Basic are either to enhance the existing commands and instructions, or to implement additional much needed features. A commercial example of this idea is Microdower's Basic Extensions ROM reviewed in BEEBUG Vol.4 No.1.

The program listed with this article implements the principles to be described and will allow further new commands and instructions to be added next month. We have also included an example this month which works with both Basic I and Basic II.

ADDING NEW COMMANDS AND INSTRUCTIONS

The principle of adding new commands is quite straightforward. When Basic encounters a command or statement which it does not recognise, it immediately jumps to an error handler which prints a suitable message. As an example, typing:

SPLODGE <Return>

in immediate mode will result in the Beeb printing 'Mistake'. If we can find a way of trapping this 'error', then we can force Basic to execute a routine of our own design implementing the new keyword.

TRAPPING MISTAKES

Normally, when Basic encounters a mistake it prints the appropriate error message and returns control to the user. At the same time it leaves a pointer (located in zero page) set to the position in the program where the mistake has occurred. Thus, to add a new command, all we need to do is to test the apparently erroneous command, to which Basic points, to see if it matches up with one of the new keywords, and then execute the corresponding new routine.

To start with, we need to decide which error to trap. The most likely error message in Basic is 'Mistake' (error number 4) and, as seen from the example above, this is printed when a single unrecognizeable word is met.

To be able to trap this 'Mistake' error, we must re-direct Basic's error vector (BRKV) which resides in page 2 of memory at &202 and &203. This initially points to the default error routine in the Basic ROM. What we must do is intercept this vector and make it jump back to our own routine when an error has occurred. If the error was not produced by one of the new keywords, then we must allow the default error routine to deal with it as usual. If the error was one of the new keywords we must act upon it. The re-directing of the error vector takes place in lines 1100 to 1230 of the program listed below.

COMMAND INTERPRETATION

To interpret the new command the 'error' is checked against a list of new keywords and, if there's a match, the relevant routine is executed. Lines 2000 to 2200 of the program provide a simple command line interpreter which handles full commands (no abbreviations allowed).

The command interpreter searches the keyword table for a keyword that matches the one producing the error. If it finds it, it reads the next two bytes from the table and uses these as a pointer to the corresponding routine (execution address). The keyword table should be structured as follows:

```
Keyword (full string)
Zero byte (end of keyword)
Execution address (two bytes, lo - hi)
Keyword
Zero byte
Execution address
   .
   .
Table termination byte (255)
```

This table is located between lines 3000 and 3500 in the main program.

BASIC TEXT POINTER

Basic has two 'text pointers' which reside in zero page. Together these point to the current place in the line of Basic. Usually, the first pointer (PTRA) is set to the start of the next statement, and the second pointer (PTRB) is used to point to the character within the statement. The two pointers are held in memory as follws:

```
PTRA     &0A   Offset
       &0B/0C  Base
PTRB     &1B   Offset
       &19/1A  Base
```

This means that when Basic encounters an error, PTRA points to the start of the statement and PTRB points to the exact position of the error in the line.

USING THE PROGRAM

The main program listed with this article provides the basic framework for implementing new keywords. This should be typed in and saved. To illustrate the technique we have added the new keyword VARS, and this appears in the keyword table (line 3000) with its corresponding routine from line 4000 up to 4210. VARS is designed to print out the values of the variables @% to Z% and although it could be included in a program is probably most useful in immediate mode. This example will also enable you to confirm that you have correctly entered the main program ready for next month's additions and extensions.

To implement the new keyword you must first run the Basic Extensions program. This installs the machine code at &900. Provided no errors occur type CALL vector <Return> and the new keyword is installed. To try out this new feature type in this short program:

```
NEW
1 REM BASEXT DEMO
2 PRINT "PRESS SHIFT FOR VARIABLES"
3 REPEAT UNTIL INKEY-1
4 VARS:PRINT "PRESS ANY KEY"
5 IFGET END
```

When you run this program you will be prompted to press the Shift key. Once you do this, the program displays the values of the variables @% to Z%, and terminates with a further key press. Notice the use of the new keyword VARS in line 4. If you press Break, the routine will no longer be enabled. To re-call the routine, type CALL &900 <Return>.

Next month we shall see how many more interesting and complex features can be added to breathe new life into the Beeb's old Basic.

```
  10 REM PROGRAM BASIC EXTENSIONS
  20 REM VERSION B0.1
  30 REM AUTHOR  Alan Webster
  40 REM BEEBUG  APRIL 1986
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60:
 100 MODE 7
 110 ON ERROR GOTO 170
 120 PROCassem
 130 PRINT "Start address &";~vector
 140 PRINT "End address &";~P%
 150 END
 160:
 170 ON ERROR OFF
 180 MODE 7:IF ERR=17 END
 190 REPORT:PRINT" at line ";ERL
 200 END
 210:
1000 DEFPROCassem
1010 R%=(?&8015)-48
1020 IF R%=1 PROCbas1
1030 IF R%=2 PROCbas2
1040 IF R%<1 OR R%>2 PRINT "Not Basic I
or II"'':END
1050:
1060 FOR PASS=0 TO 3 STEP 3
1070 P%=&900:[OPT PASS
1080:
1090 .vector
1100 LDA &202                \ Load old
1110 LDX &203              \ Error Vector
1120 CMP #start MOD 256   \ Test if set
1130 BNE notdone         \ up previously
1140 CPX #start DIV 256  \ cos we don't
1150 BEQ done          \ want endless loop
1160:
1170 .notdone
1180 STA brkv    \ Store old error vec.
1190 STX brkv+1          \ away for later
1200 LDA #start MOD 256    \ Set up our
1210 STA &202            \ Error vector
1220 LDA #start DIV 256
1230 STA &203
1240:
1250 .done
1260 RTS               \ Set-up complete
1270:
1280 .start
1290 PHA                   \ Store error
```

```
1300 TYA                \ on stack, just
1310 PHA          \ in case it's not ours
1320:
1330 LDY #0
1340 LDA (&FD),Y \ Get the error number
1350 CMP #4       \ Is it 4? ('Mistake')
1360 BEQ mistake
1370:
1380 .notours
1390 PLA                \ Pull the error
1400 TAY                \ from the stack
1410 PLA              \ and exit via the
1420 JMP (brkv)  \ default error vector
1430:
1500:
1510 .mistake
1520 LDY &A        \ Get PTRA into 7E/7F
1530 DEY           \ for using to detect
1540 TYA            \ the command giving
1550 CLC                    \ the error
1560 ADC &B
1570 STA &7E
1580 LDA &C
1590 ADC #0
1600 STA &7F
1610 \
1620 \ (&B),Y points to statement
1630 \
1640 JSR nxtwrd  \ Is command in table?
1650 BCS notours          \ No, so exit
1660:
1670 DEY             \ Reset PTRA offset
1680 TYA           \ to point to the end
1690 CLC        \ of the current keyword
1700 ADC &A
1710 STA &A
1720:
1730 PLA       \ Pull the 2 bytes of old
1740 PLA         \ error info from stack
1750 PLA      \ Pull the 3 byte RTI from
1760 PLA                    \ The stack
1770 PLA
1780 PLA        \ Pul the return address
1790 PLA                \ from the stack
1800:
1810 JMP (&7E)     \ Execute the command
1820:
2000 \
2010 \ This is our Command interpreter
2020 \ It scans our keyword table for
2030 \ a command exactly the same as
2040 \ the one which caused the error.
2050 \ If the command is found, put the
2060 \ execution address in &7E/&7F and
2070 \ return with carry clear else
2080 \ set carry to indicate no find
2090 \
2100 .nxtwrd
2110 LDX #0:.nxtl LDY #0:.nxt2
2120 LDA (&7E),Y:STA &72:LDA table,X
2130 CMP #&FF:BEQ not:CMP #0
2140 BEQ command:CMP &72:BEQ retest
2150 .nxt3 INX:LDA table,X:BNE nxt3
2160 INX:INX:INX:JMP nxtl:.retest
2170 INX:INY:JMP nxt2:.not SEC:RTS
2180 .command INX:LDA table,X:STA &7E
2190 INX:LDA table,X:STA &7F:CLC
2200 RTS
2210:
3000 .table
3010 EQUS "VARS"
3020 EQUB 0
3030 EQUW vars
3040:
3500 EQUB 255
3510:
3520 .brkv
3530 EQUW !&202
3540:
4000 .vars
4010 JSR chkend
4020 LDY #0:LDA #0:STA &72
4030 .varlp
4040 LDA &72:CLC:ADC #64
4050 JSR &FFEE:LDA #ASC("%")
4060 JSR &FFEE:LDA #32
4070 JSR &FFEE:LDA #ASC("=")
4080 JSR &FFEE:LDA #32
4090 JSR &FFEE:LDA #ASC("&")
4100 JSR &FFEE:LDA &400,Y
4110 STA &73:INY:LDA &400,Y
4120 STA &74:INY:JSR phex:LDA &73
4130 JSR phex:INY:INY:LDA #32
4140 JSR &FFEE:JSR &FFEE:JSR &FFEE
4150 LDA &73:STA &2A:LDA &74
4160 STA &2B:STY &75:JSR plnum
4170 LDY &75:JSR &FFE7:LDA &72
4180 CLC:ADC #1:STA &72
4190 CPY #&6B:BCC varlp
4200:
4210 JMP cont
4220:
8000 :]
8010 NEXT
8020 ENDPROC
8030:
9000 DEFPROCbas1
9010 cont  = &8B0C
9020 chkend= &9810
9030 phex  = &8570
9040 plnum = &98F5
9400 ENDPROC
9410:
9500 DEFPROCbas2
9510 cont  = &8B9B
9520 chkend= &9857
9530 phex  = &B545
9540 plnum = &9923
9900 ENDPROC
```

# Fleet street EDITOR

**'Read all about it' might well be an apt description for Simon Williams' review of Fleet Street Editor. How does this Mirrorsoft package compare with the AMX Pagemaker reviewed last month?**

**Product : Fleet Street Editor**
**Supplier : Mirrorsoft**
**74, Worship Street,**
**London, EC2A 3EN.**
**01-377 4837**
**Price : £39.95 (disc)**

Mirrorsoft, the software arm of Robert Maxwell's publishing group, have already released a variety of programs for the BBC micro - mostly educational games. Now they have taken a step up and produced Fleet Street Editor (FSE), a 'publishing tool' which draws on their own experience and works within the window and icon environment which is proving so popular.

The whole package is extremely well put together. It consists of an A5 ring binder with over 100 pages of manual, two discs and a function key strip. The manual contains a comprehensive tutorial section, known as the 'Guided Tour', as well as reference on each facility of the six 'departments' into which the system is divided. It's very clearly laid out with plenty of illustrations and a good index so you can dip into it as well as reading it cover to cover.


The program itself is completely disc-based and needs no extra hardware nor ROM to function. This has the advantage of versatility and low-cost, but does mean that the package is not as easy to control as with a mouse. The protection system requires you to use the disc supplied as a work disc - never the best state of affairs.

FSE is split into six 'departments'. These are the graphics library, the studio, the copydesk, page make-up, preview and print, and administration. Each handles a particular aspect of creating the pages of your publication. A page is divided into a number of 'panels', each of which is made up from graphics and text.

The graphics library is on the second of the two discs that make up the package. This consists of hundreds of detailed graphics - logos, maps, display character fonts and many other useful designs - presented as a series of screens from which you can 'pick' the items you want and transfer them to one of your page panels. They can be positioned roughly on the panel before accurate alignment in the 'studio'. For those who want to put together a publication but perhaps lack the artistic skill to draw their own graphics, this library would be extremely useful. Mirrorsoft are thinking of releasing further library discs with other ready-made designs on them.

The studio is the most complex department within FSE, and allows you to reposition your graphics, copy them, enlarge or reduce them, mirror them or zoom in and adjust them at the pixel level. You can also add line drawings using a variety of different pen styles. The zoom screen separates each pixel, displaying them as a series of boxes. This is very confusing visually and a strain on the eyes after a few minutes work.

Once you're satisfied with the graphic content of your panel, you move to the cooydesk, which lets you add text. This can be added using FSE's own rather slow text processor, or by feeding in text from a Wordwise or View file. The text can use two different fonts per panel, which may be selected from a total of six. The text automatically wraps around any graphics, and may be justified to both margins if you like. Since there's no micro-justification in the package, however, this can lead to big gaps between words.

Headlines, at twice the size of a normal character, can be added to the text, again using a selection of different fonts.

The page make-up department allows you to select the panels you've saved to disc and make up a complete A4 page. You can only have a maximum of eight panels on any page, and each is normally the same size. You can stretch a panel to twice its width, but this of course elongates any graphics by the same amount.

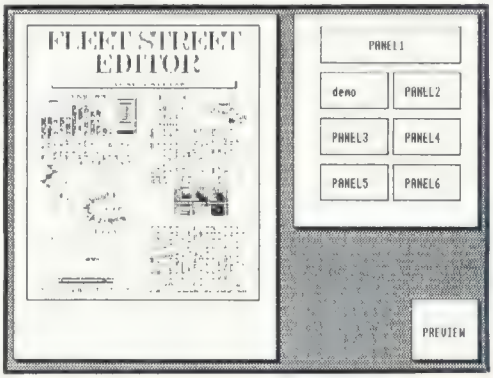
The preview and print department allows you to see a complete page on screen, at a reduced size. The text is unreadable, but you get an impression of the overall design before committing the page to paper (see illustration).

The administration department is not directly connected with the publication process, but covers orinter selection, formatting of discs (a nice touch) and the transfer of non FSE files into the system. You could use this function to import digitized pictures or (perish the thought) AMX art screens.

Comparisons will inevitably be made with the AMX Pagemaker (reviewed last month) but the packages are not as similar as might be thought. The philosophies behind the two are very different and each provides facilities not given by the other.

The biggest advantage of Pagemaker is its full page approach which allows text and graphics to be positioned anywhere on the page without the restrictions of FSE's panels. The use of the mouse and total flexibility of type sizes must also be plus points.


Against this, FSE provides an extensive graphics library, and allows you to re-edit text once it's laid down on a panel, a very useful operation that cannot easily be performed with Pagemaker. FSE is a very well thought-out and useful product and Clares Micro Supplies, who programmed it, can be justifiably proud of the features they managed to squeeze into the limited memory of the model B.



# DIRECT DISPLAY UTILITY

**You're developing a program in memory and you suddenly need to check the contents of another program saved away on tape or disc. If that sounds like trouble then this utility by Jagdish Sah could be the answer.**

Very often one wishes to list a Basic program directly from a tape or disc file in which it is stored. Perhaps you need to examine some programs without corrupting the one you already have in memory. Or you may like to copy some lines from one of your old programs into the one you are currently developing. It is very laborious to save your new program, load the old one, spool some lines from it, load the new one and *EXEC the lines in. Much easier is to have this utility to hand.

This utility will directly list to screen any Basic program, saved in the normal way (i.e. tokenized and not spooled out) from cassette or disc, for inspection or copying, without disturbing the program in memory.

Type in the program and save a copy before you run it. If it runs successfully it will prompt you to save the machine code produced as a file 'LISTMC'. From then on, whenever you wish to use the utility, just type:

```
*RUN LISTMC
```

This implements a new *LINE command. To list a Basic program from a file called (say) MYPROG, type:

```
*LINE MYPROG
```

The program will be listed in paged mode. Pressing the Shift key will proceed further with the listing. For the sake of clarity, a space is added after each line number. You could use the Shift key to find the part of the program that you are interested in, press Escape to return to immediate mode, and copy the section of the old program into your new program using the Copy key in the normal way. When Escape is pressed, or the whole file has been listed, the routine will close the file automatically and exit to Basic.

The program works with both tape and disc filing systems. If you are using tape it is advisable to issue the command:

```
*OPT 1,0
```

prior to the *LINE command to prevent the file name, block number etc. appearing together with the program listing. In this case you could also use the simplified form of the command:

```
*LINE ""
```

to list the next file on the tape.

If you press the Break key at any time the *LINE command will no longer work, but can be re-initialised by typing:

```
CALL &900
```

The program will work under both Basic I and Basic II and the RESTORE at line 1070 will choose the correct addresses for the Basic ROM routines. As it stands, the utility will not work on the Master as the ROM routines used are different in issue IV Basic. However, we hope to publish the Basic ROM addresses to enable its use on the Master as soon as we (or any enterprising Beebug member) can locate them.

Once the main program has been run and the machine code saved, you can call the utility from tape or disc at any time, using *RUN LISTMC, without losing the program in memory. Please note, that once the utility has been assembled, the machine code file cannot be used on a machine fitted with a different version of Basic. You will need to re-assemble another version for this purpose, by re-running the original program supplied here.

TECHNICAL NOTES

The program works by reading one line at a time from the program file. The line is loaded at a memory location determined by the TOP of the current program in memory. It is then listed using Basic ROM routines. The routines used are:

printchar : Prints the ASCII character in the accumulator.

ptokens : Prints the Basic keyword whose token is in the accumulator. If it is not a token it would be printed as an ordinary character, otherwise it is de-tokenized and printed as the Basic keyword.

lineno : Prints a line number.

lineno+4 : Prints the line number with a field width of 5. This is how line numbers appear when listed in Basic.

chkgoto : The line number after GOTO, GOSUB, THEN, ELSE etc. is stored in a specially coded form. This routine prints it if it is found in this form.

exit : Exits to the Basic interpreter after the machine code routine has terminated.

Refer to lines 2700 and 2730 for the actual entry points of the ROM routines for the two versions of Basic.

The utility expects each line of Basic to follow a standard format. Each line should be made up of &0D, line number (2 bytes), line length (1 byte) and Basic text. If the program finds any deviation from this format it stops listing. This avoids any random and potentially harmful consequences of using the *LINE command with a non-Basic file.

```
   10 REM PROGRAM Basic List
   20 REM VERSION 1.1
   30 REM AUTHOR  Jagdish Sah
   40 REM BEEBUG  APRIL 1986
   50 REM PROGRAM SUBJECT TO COPYRIGHT
   60 :
  100 start=&900
  110 PROCassemble
  120 PROCchecksum
  130 IF S%<>checksum%:PRINT'"Checksum e
rror"':END
  140 PRINT'''"Type"'''"*SAVE LISTMC ";~st
art;" ";~P%
  150 PRINT'"to save the machine code pr
oduced."'
  160 END
  170 :
 1000 DEF PROCassemble
 1010 linelen=&70:quoteflag=&71
 1020 savex=&72:savey=&73:handle=&74
 1030 stack=&75:lineptr=&B:top=&12
 1040 intA=&2A:osbyte=&FFF4
 1050 osfind=&FFCE:osbget=&FFD7
 1060 oswrch=&FFEE:osnewl=&FFE7
 1070 IF ?&8015=ASC("1") THEN RESTORE 27
10 ELSE RESTORE 2740
 1080 READ printchar,ptokens,lineno
 1090 READ chkgoto,exit,checksum%
 1100 FOR opt=0 TO 2 STEP 2
 1110 P%=start
 1120 [OPT opt
 1130 LDA &201
 1140 CMP #newrtn DIV 256
```

```
>*LINE B.BASES
   10 REM Program BASES
   20 REM Version 1.0
   30 REM Author Ian Waugh
   40 REM BEEBUG April 1986
   50 REM Program Subject to Copyright
   60 :
  100 MODE7
  110 :
  120 PRINTCHR$(146);" DEC";TAB(8)"HEX";
TAB(17)"BINARY"
  130 VDU28,0,24,39,1
  140 :
  150 FOR num=1 TO 256
  160 dec$=STR$(num)
  170 PRINTTAB(5-LEN(dec$));dec$;
  180 hex$=FNbase(16,num)
>
>LIST
   10REM PROGRAM Basic List
   20REM VERSION 1.1
   30REM AUTHOR  Jagdish Sah
   40REM BEEBUG  APRIL 1986
   50REM PROGRAM SUBJECT TO COPYRIGHT
   60:
  100PROCassemble
  110PROCchecksum
  120IF err% THEN END
>
```

```
 1150 BEQ out
 1160 STA oldrtn+2
 1170 LDA &200
 1180 STA oldrtn+1
 1190 LDA #newrtn MOD 256
 1200 SEI
 1210 STA &200
 1220 LDA #newrtn DIV 256
 1230 STA &201
 1240 CLI
 1250 .out
 1260 RTS
 1270 .oldrtn
 1280 JMP 0
 1290 :
 1300 .newrtn
 1310 CMP #1
 1320 BNE oldrtn
 1330 LDA #&40
 1340 JSR osfind
 1350 TAX
 1360 BNE filefound
 1370 BRK
 1380 OPT FNequb(&D6)
 1390 OPT FNequs("File not found")
 1400 OPT FNequb(0)
 1410 .filefound
 1420 STA handle
 1430 TSX
 1440 STX stack
 1450 LDA top
 1460 STA lineptr
 1470 LDA top+1
 1480 STA lineptr+1
 1490 LDA #14
 1500 JSR oswrch
 1510 :
 1520 .mainloop
 1530 LDY #0
 1540 JSR getabyte
 1550 CMP #&0D
 1560 BNE close
 1570 JSR getabyte
```

```
1580 STA intA+1
1590 TAX
1600 BMI close
1610 JSR getabyte
1620 STA intA
1630 JSR getabyte
1640 STA linelen
1650 TAX
1660 .getline
1670 JSR getabyte
1680 DEX
1690 CPX #4
1700 BNE getline
1710 JSR list
1720 CLC
1730 BCC mainloop
1740 :
1750 .close
1760 LDA #0
1770 TAY
1780 JSR osfind
1790 JSR setbasic
1800 LDA #15
1810 JSR oswrch
1820 LDA #126
1830 JSR osbyte
1840 LDX stack
1850 TXS
1860 JMP exit
1870 :
1880 .getabyte
1890 STX savex
1900 STY savey
1910 LDA &FF
1920 BMI close
1930 LDY handle
1940 JSR osbget
1950 BCS close
1960 LDX savex
1970 LDY savey
1980 STA (lineptr),Y
1990 INY
2000 RTS
2010 :
2020 .setbasic
2030 LDA #&BB
2040 LDX #0
2050 LDY #&FF
2060 JSR osbyte
2070 TXA
2080 TAY
2090 LDA #&97
2100 LDX #&30
2110 JMP osbyte
2120 :
2130 .list
2140 JSR setbasic
2150 LDY #0
2160 STY quoteflag
2170 JSR lineno+4
2180 LDA #32
2190 JSR oswrch
2200 LDY #4
2210 .printloop
2220 LDA (lineptr),Y
2230 CMP #ASC("""")
2240 BNE notquote
2250 EOR quoteflag
2260 STA quoteflag
2270 LDA #ASC("""")
2280 .notquote
2290 LDX quoteflag
2300 BNE inquotes
2310 JSR chkgoto
2320 BCC notgoto
2330 STY savey
2340 JSR lineno
2350 LDY savey
2360 BNE check2
2370 .notgoto
2380 JSR ptokens
2390 CLC
2400 BCC check
2410 .inquotes
2420 JSR printchar
2430 .check
2440 INY
2450 .check2
2460 CPY linelen
2470 BNE printloop
2480 JMP osnewl
2490 ]
2500 NEXT
2510 ENDPROC
2520 :
2530 DEF FNequb(byte)
2540 ?P%=byte
2550 P%=P%+1
2560 =opt
2570 :
2580 DEF FNequs(A$)
2590 $P%=A$
2600 P%=P%+LEN(A$)
2610 =opt
2620 :
2630 DEF PROCchecksum
2640 S%=0
2650 FOR J%=start TO P%-1
2660 S%=S%+?J%
2670 NEXT
2680 ENDPROC
2690 :
2700 REM Basic-I entry points and check
sum value
2710 DATA &B571,&B53A,&98F1
2720 DATA &97B6,&8A99,28932
2730 REM Basic-II entry points and chec
ksum value
2740 DATA &B558,&B50E,&991F
2750 DATA &97E7,&8AF6,28587
```

# Micro PROLOG

**Prolog is one of the new artificial intelligence languages, now all the rage with computing afficionados. Mark Sealey introduces the language, and reviews Acornsoft's recent release of Prolog for the Beeb.**

**Product : Micro Prolog**
**Supplier : Acornsoft,**
**Cambridge Technopark,**
**645 Newmarket Road,**
**Cambridge CB5 8PD.**
**0223-21441**
**Price : £79.95 (recommended retail price)**

England win first-test
West-Indies lose in-Kingston

and only the hyphens to say it isn't true! If you've come to computing through Basic, Pascal or even Logo, not to mention assembler or one of the older languages like Fortran or Cobol, it will seem inconceivable to you that the above two statements could be a computer program. Yet they are. Micro Prolog allows just such unlikely events to be handled with as much ease and fun as:

```
5 LET A% = 10
10 PRINT 34 + A%"
```

Micro Prolog is the latest language to be released for the BBC micro by Acornsoft and is unlike any other language for the Beeb. For that reason we will take a look at the Prolog language itself before considering this implementation. Prolog allows the computer to store statements and models of the real world as text, in a variety of forms. Prolog is then able to act upon these in a strictly logical way - Prolog means PROgramming in LOGic. Prolog is the language of expert systems and so-called 5th generation computers.

THE PROLOG LANGUAGE

Prolog's core unit (roughly akin in simplicity to something like lines 5 and 10 above) is the 'atomic' sentence:

Botham likes hitting-sixes
England won the-ashes
Penny likes Paul

What this does is establish a RELATIONSHIP (the essence of the atomic unit) between Botham and hitting-sixes, or Penny and Paul, etc. The computer will in future recognize this relationship by the term "likes". Similarly the word "won" will identify the standing of "England" to "the-ashes". The hyphen is necessary to tell the computer to treat "the" and "ashes" as one object in the sentence.

Prolog is not a language dealing with a sequence of instructions as Basic or Comal is. It is a descriptive language, and many of its statements appear to do little more than state possible truths about two or more objects (or "terms" as they are more properly called in Prolog). Assigning values to variables as in Basic is rather foreign to Prolog. Nor do programs distinguish between data types, or even between data, as such, and the program. Not to understand these points will hold up the intending user and lead to real frustration.

It is possible to enter as many facts as you know about cricket, for instance, or who likes whom in a group of people, and then interrogate the program as you would a conventional database.

## Micro PROLOG

Prolog allows a variety of "query" statements to get at logical links. They take one of several forms:

is (Botham likes hitting-sixes)

and the computer will reply with a:

Yes

to show that the inquiry resulted in a positive match and that you had previously said that he does. The only other possible outcome was a negative one. Alternatively, use a variable (here it is x) to find which of a number of relationships stored is true. Remember the basic form is: INDIVIDUAL..RELATIONSHIP..INDIVIDUAL. In English you would ask, "Who is it that likes hitting sixes?" In Prolog the form (strange at first) is:

which (x: x likes hitting-sixes)

reply:

Botham

If you had entered the names of many more sloggers like that, they would all be recalled one by one. But, having now used the word, it is possible to define for the Prolog environment what "slogging" is. This uses a conditional:

x is-a slogger if x likes hitting-sixes

note the hyphens and try to divide the sentence into its two INDIVIDUAL.. RELATION..INDIVIDUAL components, joined by "if". Prolog doesn't know or care that "is" and "a" are two words to us, nor that "likes" has previously been used in another context. All Prolog has done is associate every existing and future occurrence (there are no line numbers or loops in Prolog) of an individual who is distinguished by liking to hit sixes with the term "slogger". Thus the query

which (x: x is-a slogger)

will throw up Botham again - and as many like him as you have entered!

## *Micro* PROLOG

But we should be more subtle. Prolog uses AND, OR etc common to almost all programming languages:

x is-a reckless-slogger if x likes hitting-sixes and x takes risks

For Botham to qualify we should add:

Botham takes risks

The possibilities are infinite. We could now extend the program to include all those cricketers who took risks. Note, though, that although we slip easily from one syntactical form to another in speaking - "take" or "took" is the same root - Prolog expects an identical spelling if it is to find matches.

By compiling sophisticated sets of truths and relationships it is soon possible to design a query program that will pull out the obscurest and apparently most concealed of facts about the data entered. You will have begun a foray into the world of AI (Artificial Intelligence).

Space does not allow me here to look in greater detail. Suffice it to say that I find it exciting and refreshing to be able to work almost entirely in words and sentences when programming. I have come to prefer working this way in Prolog to either Logo or Lisp. Nor must it be forgotten that Prolog can handle numbers too, and the Beeb implementation uses both integers and floating point numbers, and far too many other features (like recursion, file handling and a host of ways of arranging data once entered) to go into here!

If you want to learn structured algorithmic techniques or interface your machine to external devices, forget Prolog. It is a language favouring the solutions to tasks where the computer handles the analysis, not the programmer, and where even complex data organisation is easy. If you are keen to build up an understanding of how a computer can simulate human "intelligence", work in a semantic structure close to the one we use every day, manipulate truths, logic and even language itself, then you should investigate Prolog further.

ACORNSOFT PROLOG

Logic Programming Associates (who have written Micro-Prolog not only for Acorn but for several other popular micros) follow the standard practice of adding up to three enhancements to the Prolog idea. These are known as "front-ends". For newcomers to the language, it is enough to say that SIMPLE, MICRO and MITSI allow entry of the atoms and more complex sentence forms in a variety of slightly different ways, using a more accessible syntax. With these Acornsoft have given newcomers a head's start, although with the inevitable penalty of some loss of user program space and power. My suggestion for them would be to get to grips with the sort of clauses looked at earlier and then explore the rules specific to each front-end with the aid of the Acorn book, only later moving back to 'pure Prolog'.

## *Micro* PROLOG

I found all three additions to the top level Prolog Supervisor easy and reliable to use. Given the range of extensions, the authors have gone a long way to de-mystify what might otherwise seem an off-putting new venture. For example, the command 'accept', which allows you to enter as many individuals as you want against a single prompt of the common relation between them all:

accept likes

lets you get away with just "Penny Jane", "Penny Emma" and "Penny Nadia" (standing for "Penny likes Jane" etc). While 'likes' is not a long word, imagine entering details of drugs in a micro expert system:

accept has-the-side-effect-of

would save a lot of typing.

You can remind yourself whether anybody likes anybody at all by using the 'defined' function. Ask:

is (likes defined)

and get the answer Yes or No. Further commands allow many different sets of data to be sifted, making checking, adding and modelling pleasingly straightforward.

Acorn has scored throughout on ease and flexibility of use like this. There is a simple and effective suite of commands to delete, list and edit your programs. This all increases the feeling of being in control. Many of these overlays can be deleted from memory and the space reclaimed once they are no longer needed.

On the other hand, to have so many commands and utilities loaded in from disc (almost like library routines in BCPL or Pascal) does tend to give the language a slightly artificial feel. This is aggravated by the impression that areas of memory (especially in MITSI, MICRO and SIMPLE) are furiously and jealously jostling for supremacy. You cannot change mode, for instance, if this would corrupt your program. Yet you are not so coddled as in Acornsoft's disc Lisp, and there is also a HiLog version for Second Processor users that overcomes many of these restrictions. Prolog also runs on the B+ by the way.

## *Micro* PROLOG

I did not find the work I wanted to do was in any way hampered by lack of room. I imagine that most Prolog owners will want just to experiment in the first place anyway. Operation here is relatively slow and only a scaled down expert system (allowing representation and modelling of knowledge in such a way that the computer appears to give 'intelligent' answers when consulted) could be implemented. After loading MICRO, there is just 12K of program space and only 8K with MITSI. But consider the delight with which younger users will be able to type in:

I can bat.

someone can play-cricket if someone can bat.

anyone can play-cricket?

and the reply:

I can

Note the precise punctuation. No x or y here and the query "why" can even be incorporated - a very approachable start and a great way in to artificial intelligence, one which for absolute beginners and children I would recommend over SIMPLE.

Back in pure Prolog, even greater things are possible. You can define your own commands from relations that you have set up. Like Forth, Prolog extends itself though I felt the book was obscure on this point. Your programs can 'learn' new facts like the instructive prime and factor demonstration program that comes on the disc. More of these examples would have been welcome.

Although Prolog cannot do wonders with graphics, all BBC graphics modes are supported and Acorn have gone to their usual admirable lengths to provide information on and access to O.S., VDU, OSBYTE and OSWORD commands. ADVAL, SOUND and ENVELOPE (usually with a list or block as parameters) are supported and I found they worked well.

The front-ends supplied not only have built-in, clear and quite helpful error messages (especially MICRO) but also modules that allow tracing and 'spying' as well as error-trapping routines of your own design.

On the minus side, filenames can NOT have up to 16 characters as an Appendix states; there is a bug in the spelling on screen of 'sentenceform', and the "edit!" command in MITSI failed to work. These would indicate oversights that were not at all typical of the package as a whole. It IS an expensive addition to your collection of ROMs but one which, because of its flexibility, reliability and dissimilarity to any other language, I can strongly recommend.

SUGGESTED READING:

"Beginning Micro-Prolog", Ennals, Horwood.
"LPA Micro-Prolog Primer", Clark, Ennals, McCabe, Imperial College/LPA
"Programming in Micro-Prolog made Simple", Hepburn; Horwood
"Prolog and Artificial Intelligence", Berk, Collins

# Computer Tools for the Blind

**A year ago, David Calderwood described some of the ways he had adapted computer games for the blind. Now he reports on some of his recent experiences with more serious applications.**

About a year ago (BEEBUG Vol.3 No.8) I reported to readers some progress in writing or modifying computer games to make them playable by blind Beeb users. Now I want to concentrate on the other side of the coin - using the micro as a tool. Several custom made software packages exist using a Votrax type speech synthesizer as an output device. Dr. Tom Vincent and Steve Turnbull of the Open University have produced a Talking Word Processor and a Talking Data Base amongst other talking utilities. However, more recent packages from other sources are getting to be very expensive.

The ideal solution would be to use existing software written for sighted use. Alas, this is almost always useless - the presentation of a screenful of information in spoken serial form is enough to drive anyone to start biting the keys! Paul Blenkhorn of Birmingham University made steps towards solving this problem: he wrote a Screen Reader. This little program lifted PAGE and lived in the space thus provided - the user then had a normal Beeb with slightly less memory. A sighted program could then be loaded and run. The blind user could then examine the screen by hitting the tab key to change the keyboard into screen reading mode - a line or word or even a single character could then be read, the cursor moved around the screen and a touch on the X or Y key would give the cursor's coordinate. When the desired information has been gleaned the Return key is pressed to return the computer to its normal mode of operation.

So far so good - however the Screen Reader did use some memory and if the other program wanted that, then the results were predictable! Fortunately 1985 brought with it a proliferation of sideways RAM boards at reasonable prices - the answer was to give the Screen Reader a home of its own - a nice little 6264 LP chip. In that location it could read and write to its heart's content without affecting the main computer memory. The firmware packages that exist at present are the standard Screen Reader plus a modified version which works with Wordwise and View - these are the work of Paul Blenkhorn. Steve Turnbull has also written a screen reader "Microspeak" which carries with it a massive exception table which stops the synthesizer trying to pronounce collections of letters like "LDA" or "CLS"; instead it speaks individual letters.

I have spent some time trying out these programs and have been very pleased with the results - for example they work well on the disc-based Spellcheck and the recent BEEBUG Filer database (Vol.4 Nos. 6 - 8). On the leisure side a blind person can now play White Knight Chess, and find out the exact spelling of a secret word scratched on a cave wall in an adventure (XYZZY pronounced as a word sounds very funny).

Of course problems still exist - many programmers disable any key not used in their software thereby making it impossible for use by the blind. I suppose it would be too much to ask that, say, Ctrl-A could be left for use by us?

Applications for screen reading are legion, already allowing the blind access to both Teletext and Prestel via standard communications software for sighted users - the only modification necessary is to interface the synthesizer using the parallel port (the serial port is in use by the modem).

A less "high tech" communication boon, which a talking computer gives, is to assist correspondence between blind and sighted people - the blind tend to communicate between one another either by Braille or cassette: the former is difficult for a sighted person to master and the latter often induces extreme embarrassment - nothing like the sound of your own voice to bring you down a peg or two! However, a sighted Beeb user can type

a letter using a word processor and save the ASCII file onto disc or tape - a blind user can then listen to the same letter using synthesized speech. It would be wonderful to be able to access printed material such as magazines in this way but there are problems of copyright.

Research has already started on producing an audio representation of graphical information such as a trace on an oscilloscope. Perhaps it is worth mentioning cost. For years technology for the blind could be bought at a price but a very high price. The reason for this, it is claimed, is the small demand for a customised product. With the current trend towards using standard hardware and software the price is brought down to a realistic level and the technology becomes available to most. The future is promising with Braille printers just around the corner; and with other tasty devices such as character recognition cameras already in existence, the micro is likely to spread its influence in both work and leisure.

The author runs the VDU group (Visually Disabled User group). Anyone interested in further details should send a good quality C90 tape, and postage-paid self-addressed packaging, to him for a demonstration of the system.

David Calderwood, Wenallt, Harlech, Gwynedd LL46 2UE.

Other useful addresses:

Dr. Tom Vincent,
I.E.T.,
Open University,
Walton Hall,
Milton Keynes.
MK7 6AA.

Paul Blenkhorn,
Research Centre for
the Education of the
Visually Handicapped
Selby Wick House,
59 Selby Wick Road,
Birmingham B29 7JE.

# COMPETITION COMPETITION COMPETITION

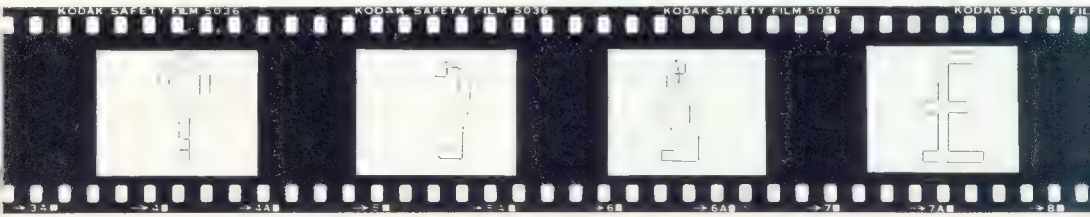
This month's excellent Inbetweening program (see page 8) gives an impressive display of contorting shapes transforming themselves into one another. In fact, it is so good that we think you ought to try your hand at designing your own inbetweening sequence - with suitable incentives, of course.

We are offering a grand prize of a voucher for £50 against any products sold by BEEBUG Retail (see the price list with this issue) for the inbetweening display sequence that is (in the opinion of the inevitable judges) the most visually attractive and entertaining.

All you have to do is to create your own DATA statements for the inbetweening program that will produce a sequence of at least four shapes. Post a cassette or disc of the complete program (along with a suitable S.A.E. if you want your cassette/disc returned) to arrive before May 16th to:

Beebug Inbetweening Competition,
PO Box 50, Holywell Hill,
St.Albans, AL1 3YS.

The lucky winning entry will be published in the July issue of BEEBUG.



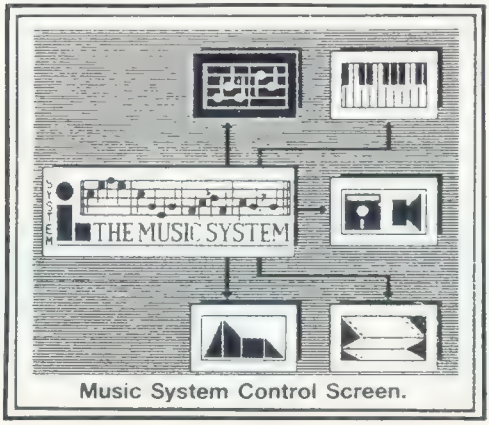
# MUSIC SYSTEMS UPDATE

**This month, BEEBUG brings you up to date on the latest hardware and software developments designed to help you make music on your BBC micro. Ian Waugh conducts.**

In BEEBUG Vol.4 No.1, we looked at the Island Logic Music System, the Clef Computer Music System, ATPL's Symphony Keyboard and Acorn's (or Hybrid's) Music 500. Since then, new utilities and Song and Sound Library discs have been produced for The Music System and two MIDI interfaces have appeared too; not to mention BEEBUGSOFT's own MUROM and Studio 8. Here we will see what The Music System extras and MIDI has to offer the budding Beeb musician.

**System, 12 Collegiate Crescent, Sheffield, S10 2BA. 0742-68321.**

The Music System (TMS) at £29.95 is undoubtedly a most sophisticated music editor. The program comes supplied with a Song and Sound Library containing examples of songs and sounds (what else?) which you can use yourself in the program. Five other Song and Sound discs have been produced by independent programmers and these are available at £4 each. They include, Toccata and Carols, 400 Years of Music, Mainly Bach, Ian Waugh Originals and Old Favourites. (Yes, that's me. Well, I thought the program so good that I wrote some pieces for it).



Music System Control Screen.

System has also released a Utilities Disc at £6 which includes a program to play TMS files from Basic. The Music System uses different envelopes and music protocol to the Beeb, so this is not easy. Before it can be attempted the TMS file must run through a converter. Every envelope and amplitude may not be converted exactly and the tempo may need adjusting. The noise channel is not converted at all, nor can it handle certain notes, but it tells you which ones so you can alter them. In spite of this, the ability to play tunes in your own Basic programs will appeal to many users.

There is also a utility to convert a keyboard file to a music file. Again, it does not promise to convert it exactly but it does the best it can. A third converter is for owners of System's Music Editor, the forerunner to The Music System, and this converts ME files to TMS files.

The Utilities and Song and Sound discs are only available on disc from System by mail order (or at computer shows).

**Electromusic Research, 14 Mount Close, Wickford, Essex, SS11 8HG. 0702-335747.**

On to something a little heavier now - MIDI - and a few words of explanation for anyone to whom this is unfamiliar. MIDI is an acronym for Musical Instrument Digital Interface and was developed to allow different electronic instruments to communicate with each other. Typically, one synthesizer can be used to control others in synchronization with a sequencer and drum machine.

As MIDI uses digital signals it is a prime target for computer control. A MIDI interface bridges the hardware gap between the computer and external instruments and the software allows you to compose a piece and play it back on one or more instruments. There are 16 MIDI channels

three other packages and updated the original. They fall under the general title of Miditrack and include the Performer (£49.95), the Composer (£44.95), the Music Editor (£39.95) and Vu-Music (£24.95). EMR's MIDI interface itself costs £79.90.

There are two ways to put music into a MIDI system: real-time and step-time. Real-time music is recorded from a keyboard 'live'. Step-time music is entered one note at a time from a music keyboard or from the computer. Although real-time is great if you are an accurate player, step-time lets you program pieces which are impossible to play, and with 100% accuracy. The Performer is a real-time system and the two demo programs on the disc - an excerpt from the fourth movement of Beethoven's Sonata in C minor and some jazzed-up Christmas carols - show what can be achieved.

You can record on up to eight tracks, working in a similar way to recording on an eight track tape recorder, but you can transpose a track up or down 12 semitones and cause it to loop continuously. Performance information, such as pitch bend, can be recorded too. A polyphonic and mono mode allows you to use multi-timbral instruments (which can sound more than one voice at the same time) such as Casio's CZ-101 and CZ-1000. These are a cheap and effective way of using MIDI's multi-tracking ability. Unfortunately, the metronome sends its 'tick' through channel 1 which affected the Casio when in multi-timbral mode. A tick through the Beeb's speaker would have been better.

You can create 64 arrangements of the recorded tracks and an auto note-correct feature (often called 'quantization') will correct any small timing mistakes you may make. By routing the output to the input you can also merge tracks.

The Performer is reasonably easy to use in spite of its wealth of facilities, but it does take a while to get to grips with and the program crashed once in the process. The only editing facility is a punch-in system but the Music Editor was designed for that purpose.

The Composer is menu driven and gives you six tracks to record on. You can allocate MIDI channel numbers to the tracks and set Mono or Polyphonic mode. Multi-timbral instruments perform superbly.

Notes can be entered from an instrument or the computer. They are shown as a series of numbers representing such parameters as pitch, duration and volume. It can be more difficult to read than traditional notation but it has advantages and it soon becomes familiar. You can insert and delete notes and rests, alter the voices, change dynamics and add modulation control. A function key strip helps. You can also copy sections to other tracks with optional transposition.

The Music Editor complements the Performer and the Composer. It converts Composer files to Performer files and allows full editing of a polyphonic track. Each byte of MIDI information is shown and you cannot get much further into the system than that. Full editing is therefore possible, once you have come to grips with the numeric notation. Demos are included for you to sift through and learn from. Such delvings are not for the faint-hearted but they do improve your ultimate control over compositions.

Files can be linked for continuous playback and free memory has been increased by 50% over the Performer program. Another program called Music Player loads and plays Performer files.

Vu-Music produces a graphic display which varies according to the music played. Input is only in real-time via a MIDI keyboard and you cannot play

pre-recorded pieces, which is a pity.

EMR are also working on The Notator which should retail for £34.95. This will print out a track from the Composer in traditional music notation.

For MIDI software, these programs are cheap and they offer a lot of facilities not always found in MIDI packages. They may take a little getting used to, but as your music system grows, it should keep pace with you.

**The London Rock Shop, 26 Chalk Farm Road, London, NW1 8AG. 01-267 7851.**

From one end of the MIDI interface spectrum to the other - the UMI-2B retails for £495. It is probably the most expensive MIDI interface system on the market even though you get the hardware interface and software (on EPROM) for your money.

The box has enough plugs and sockets to ensure compatibility with just about anything. Note entry is really geared to real-time but it does have a step-time facility too. The program is menu driven from a series of mode 7 screens.

Recording is as simple as pressing a button and playing some notes. The sequence will automatically play back so you can develop other lines to go with it. It is really the ideal tool for the songwriter or jingle writer and many professionals are already using it.

Each sequence you record is called a pattern and up to four will play back on auto replay. Patterns can be linked together to form a song and saved to disc together with a notes page onto which you can enter details of the recording.

The program remembers all modulation controls and, as these use a lot of memory, a pack function will reduce them bit by bit until they nearly disappear. If you go too far, all is not lost as you can call up the original setting again. There is also an auto note correction facility and, as with 'pack', you can return to the original, too.

Step-time and real-time note entry can and instruments can be instructed to send or receive information on any channel.

To produce MIDI music with your Beeb you will need a MIDI interface, a synthesizer, or other instrument with MIDI connections, and suitable software. Most of this turns your MIDI/micro system into a super sequencer or a digital tape recorder - only better, as pieces can be speeded up, slowed down, and individual notes and phrases very easily edited with no loss of timing or signal quality.

EMR have produced more MIDI packages for computers than any other company and the BBC micro ranks high on their list; in fact they were the first to produce a MIDI package for the Beeb. That was almost two years ago and they have since produced be combined so you can enter difficult sequences easily. The program also includes a patch and bank dump for Yamaha's DX7 synth.

This is really a professional package aimed squarely at the professional musician, not the computer buff. It has, unfortunately, a professional price tag to match, but if you can afford it, it is a very useful piece of equipment.

Music, and MIDI-controlled music especially, is a complex subject at the best of times. However, a micro can do much to alleviate this and change the skills needed of a musician from pure technique to a more artistic bias. Indeed, you can now have next to no technical skills at all and still produce good music. With the products I have looked at here, owners of a BBC micro will find help on this score at all levels.

# 1st course

## Introducing Hexadecimal

**If you class yourself as something of a novice when it comes to computing, Ian Waugh's gentle introduction to the hexadecimal world may be just what you need.**

Hexadecimal (hex for short) is one aspect of computing which can look pretty off-putting at first sight. The mixture of letters and numbers looks more like a code than the counting system it is.

Hex is not difficult to understand but before you bother to learn what it is, you may well want to know why you should bother with it at all. The simple and most practical answer is so that you will be able to understand those articles and programs which use hex - and programmers use hex for only one reason - it makes programming a little easier. If this seems like a contradiction, read on.

Hex is often used as a halfway house between binary and decimal. Computers work mainly in binary which is fine for THEM but rather difficult for US. Decimal uses a base of 10, binary uses a base of 2 and hex uses a base of 16. As we do not have a single character to represent numbers greater than 9, hex uses the letters A to F for the values 10 to 15. The ampersand (&) is used to tell the computer that what follows is a hexadecimal number and not a decimal number or variable.

Figure 1 lists some numbers in the three counting systems. You can see, for example, the difference between 10 (decimal) and &10 (hex).

In decimal when we move one column to the left we multiply by 10, in binary we multiply by 2 and in hex we multiply by 16. The column headings in hex are as shown in figure 2.

Now, the 16 times multiplication table is not on my list of favourite subjects but there is no need to learn it as the Beeb has a built-in converter. To convert hex to decimal just enter the hex number with the ampersand, for example:

```
PRINT &EA
```

will result in 234. The tilde (~), on the exponential key, will print a decimal number in hex, for example:

```
PRINT ~234
```

will result in EA being displayed. Note, the computer does not insert an ampersand; it assumes you know what you asked it to do. A hex number without an ampersand can be evaluated using the EVAL function:

```
10 INPUT "Hex number" hS
20 PRINT "Decimal = ";EVAL("&"+h$)
```

| Decimal | Binary | Hex |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 10 | 2 |
| 3 | 11 | 3 |
| 4 | 100 | 4 |
| 5 | 101 | 5 |
| 6 | 110 | 6 |
| 7 | 111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |
| 16 | 10000 | 10 |
| 32 | 100000 | 20 |
| 255 | 11111111 | FF |
| 256 | 100000000 | 100 |

Figure 1

Almost everyone with a disc interface will have used hex in a practical way. Most readers will know that Basic programs start from memory location &E00 in cassette-based machines and &1900 in those with an Acorn disc interface. If you have

| | 16^4 | 16^3 | 16^2 | 16^1 | 16^0 |
|---|---|---|---|---|---|
| Hex | 10000 | 1000 | 100 | 10 | 1 |
| Decimal | 65536 | 4096 | 256 | 16 | 1 |

Figure 2

a disc based machine and want to load a long tape program you will normally enter:

```
PAGE=&E00
```

before loading the program. You could just as easily enter:

```
PAGE=3584
```

which is &E00 in hex, but is not &E00 far easier to remember?

PAGE is a pseudo-variable and gives the address in memory where Basic stores or loads your program. The Beeb always sets PAGE to a multiple of &100 so the difference between one PAGE and the next is 256 (&100) bytes.

Pages, therefore, can be used as a convenient measure of memory. It is far easier to add several pages in hex than decimal. Quickly, how many bytes is eight pages? You can do it the long way and say 8 x 256 = 2048 or you can do it the easy way and say 8 x &100 = &800. You do need to learn that A=10 etc., so that 10 pages equals &A00 but that is not too difficult.

You may have realised that the number 8 and its multiples appear very often in programming especially when used to describe memory locations. Take a look at the memory map on pages 500 and 501 of the User Guide (pages 493 and 494 of the new User Guide) for an excellent example and comparison between hex and decimal numbers. See how easy it is to load function key definitions:

```
*LOAD KEYS B00
```

and machine code programs into sideways RAM:

```
*LOAD MYPROG 8000
```

An 8K ROM is &2000 bytes long and a 16K ROM holds &4000 bytes. Far easier to remember and manipulate than 8192 and 16384. When you *SAVE a section of memory, the computer expects the start and end addresses (or length in bytes) in hexadecimal. You do not use the ampersand as the computer is expecting hex. In fact, if you DO use an ampersand the computer will report an error. For example, you can save a 20K screen display in mode 0, 1 or 2 with the following instruction:

```
*SAVE SCREEN 3000+5000
```

Many ROMs, especialy toolkit ROMs, expect parameters to be given in hex simply because when you are messing around with chunks of memory, hex is usually easier to work with, although it may take a little getting used to initially.

I said hex was used as a halfway house between binary and decimal and this can be most useful when working at bit level. Bits are not just the concern of machine code programmers. The arguments of several Basic commands are not interpreted by the computer as a decimal or hex value but as a string of binary digits. For example, look at the use of @% in PRINT format control (see First Course in BEEBUG Vol.4 No.5 and No.6) and the expanded Channel parameter of the SOUND command (see BEEBUG Vol.3 No.9 p.27). Try working with these commands in decimal and you will see how difficult it is.

When working at bit level, we are concerned not with the value of the byte (in hex or decimal) but with the bit pattern, i.e. the byte in binary. As the Beeb is an 8-bit micro, the largest value a byte can hold is 255 or 11111111 in binary and &FF in hex.

There is an easy way to convert between binary and hex. This is to split the binary number into two sections of 4-bits (called a nibble, naturally) and replace each section with its hex equivalent. For example 10011101 can be split into 1001 and 1101 which is 9 and D respectively so &9D is the hex equivalent of 10011101. A more practical use, perhaps, is converting from hex to binary as this lets you see the bit pattern of a number fairly easily. A decimal number gives no indication at all.

Bit patterns are useful for all sorts of things as you can store eight items of information in a single byte: a very efficient use of memory. One application would be in an adventure game where a byte could hold the exits of a room. For example, 11010100 could mean (reading from left to right) there are exits North, East, not South, West, not Up, Down - and you still have two bits left over. In hex this is &D4 and it is easier to convert this to binary to see the bit pattern than to try to convert 212, the decimal equivalent, which gives no hint as to the bit pattern at all.

The interpretation of a number as a string of binary digits rather than as a value is used a lot in control applications too, where each bit will

turn a piece of equipment on or off. If you study the screen and pixel displays of the graphic modes you will see how bit patterns are useful here, too (see Advanced User Guide page 462).

Some computers allow you to work in binary and have binary conversion functions but as we cannot do this on the Beeb we must work in decimal or hex, and hex, as I hope you have seen, makes it easier to visualise the bit pattern.

To carry our adventure scenario a step further: the room exits would be stored sequentially (in an array or area of memory) and an exit could be tested for by ANDing with a specific value. For example to discover if the room has an exit East, AND its byte with %01000000 or &40:

```
PRINT &40=(&D4 AND &40)
```

The result will be -1 which is TRUE. In a program, this would be turned into a function which could test for any exit in any room. (See BEEBUG Vol. 4 No. 2 and No.3 for more information about logic and the use of AND.) You can see how difficult it would be to design such a routine using decimal numbers.

So, hex has its place in the computerised scheme of things. If you progress to machine code, a knowledge of hex is essential. Until then, an understanding of hex will increase your understanding of the way your computer operates and it can be a great help with several aspects of programming.

As an illustration of converting numbers between bases, I will leave you with a program that you may like to try out.

The following program will count from 1 to 256 in decimal, printing the equivalent hex and binary values. The principles of converting a number from one base to another are quite straightforward, although the process can appear a little confusing at first sight. FNbase converts a decimal number to another base and with a little more work it would be possible to rewrite the program to convert from any given base to another. Decimal, hex and binary will be all that 99.9% of us ever need 99.9% of the time, so I will leave further tinkering to the other 0.1% of you.

THE PROGRAM ALGORITHM

The multiples of the base are subtracted from the number to be converted and the remainders become part of the new number. After each subtraction, the figure is given the value of the last multiple. This can be seen in the function. If you are unfamiliar with MOD and DIV, line 1070 may be easier to understand if written as follows:

```
1070 mult=INT(fig/base):rem=fig-base*mult
```

```
   10 REM Program BASES
   20 REM Version 1.0
   30 REM Author Ian Waugh
   40 REM BEEBUG April 1986
   50 REM Program Subject to Copyright
   60 :
  100 MODE7
  120 PRINTCHR$(146);" DEC";TAB(8)"HEX";
TAB(17)"BINARY"
  130 VDU28,0,24,38,1
  140 :
  150 FOR num=1 TO 256
  160 dec$=STR$(num)
  170 PRINTTAB(5-LEN(dec$));dec$;
  180 hex$=FNbase(16,num)
  190 PRINTTAB(11-LEN(hex$));hex$;
  200 bin$=FNbase(2,num)
  210 PRINTTAB(23-LEN(bin$));bin$
  220 NEXT
  230 END
  240 :
 1000 DEFFNbase(base,fig)
 1010 num$=""
 1020 :
 1030 REM DIVIDE NUMBER (fig)
 1040 REM BY BASE (base)
 1050 :
 1060 REPEAT
 1070 mult=fig DIV base:rem=fig MOD base
 1080 :
 1090 REM SET NUMBER (fig)
 1100 REM EQUAL TO MULTIPLIER (mult)
 1110 :
 1120 fig=mult
 1130 :
 1140 REM REMAINDER (rem) IS PART OF
 1150 REM NEW NUMBER (num$)
 1160 :
 1170 IF rem<10 num$=CHR$(48+rem)+num$
 1180 IF rem>9 num$=CHR$(55+rem)+num$
 1190 :
 1200 REM DO IT UNTIL NUMBER (fig) = 0
 1210 :
 1220 UNTIL fig=0
 1230 =num$
```

# Getting a Better VIEW (Part 2)

**Chris Watts concludes his look at View by showing how the function keys can add even more to your word processing.**

Last month we looked at a variety of ways in which we could improve our use of View, Acornsoft's well established word processor. Now we will take a look at perhaps the most useful addition of all, the programming of the function keys to provide many additional features.

## DEFINING FUNCTION KEYS

One of the criticisms often made of View is that the function keys cannot be defined at all by the user. This is not so. The Ctrl-Shift function keys are not used by View and may be programmed as usual. Once enabled, (with *FX228,1) they can be used in both command and text mode.

There are at least three possible ways in which you might use these keys:

+ As a shorthand for frequently used phrases, in text mode, e.g. *KEY0 Beebug,|MP.O. Box 50,|MSt Albans,|M Herts|M

+ As a shorthand for frequently used commands, in command mode, e.g. *KEY0 PRINTER FX80

+ By programming them to produce a sequence of commands in text mode.

The last of these three options is the most useful, but the most complex. To make use of it, it is necessary to know the codes generated by each of the function keys, when used by View, and to know the sequence required to program a function key to produce the appropriate code. These codes are given in Table 1.

Using these one may program the Ctrl-Shift function keys to provide additional facilities. For example, I use two sets of key definitions, one for filing/printing etc, the other for editing. These two sets of definitions are given at the end of this article, though you could just as well design your own.

## CREATING KEY DEFINITIONS

To set up these key definitions, create a Basic program, with line numbers as usual, and made up as follows (leave out all the explanatory comments given with the key definitions):

```
*FX228,1
<Key definitions 1>
*SAVE KEYDEF1 B00+FF
<Key definitions 2>
*SAVE KEYDEF2 B00+FF
END
```

Now run this program to create the two key definition files (called KEYDEF1 and KEYDEF2). *FX228,1 is needed to enable the Ctrl-Shift function keys. Other key definitions could be created in the same way.

## INSTALLING THE KEY DEFINITIONS

The best way of loading the key definitions for use is probably by including a *LOAD command in a boot file as described last month, but any set of key definitions similarly created may be loaded from within View by typing:

```
*FX228,1
*LOAD <name>
```

where <name> is the appropriate name of the *SAVEd definitions.

Once you have experimented with the key definitions as listed you may well wish to modify these to suit your own requirements. Simply edit the Basic program containing the key definitions and re-run the program to produce the new versions of the key definition files.

Note that as listed, each set defines f0 to load in the alternative set of definitions when required. The functions of the other key definitions should be clear from the accompanying comments.

On the Master 128, function key definitions are hidden away in private RAM and cannot be readily *SAVEd or *LOADed. As an alternative, the definitions alone could be created as an ASCII file (use View for this) and *EXECed in from View as required. You will also need to include *FX228,1 at the start of each file. The *L. in the definition of f0 in each case should also be replaced by *EXEC.

VIEW FUNCTION KEY CODES

| Key | Meaning in text mode | Code prod. | Code for *KEY |
|---|---|---|---|
| f0 | Format Block | 140 | \|!\|L |
| f1 | Top of text | 141 | \|!\|M |
| f2 | Bottom of text | 142 | \|!\|N |
| f3 | Delete End of Line | 143 | \|!\|O |
| f4 | Beginning of Line | 144 | \|!\|P |
| f5 | End of Line | 145 | \|!\|Q |
| f6 | Insert Line | 146 | \|!\|R |
| f7 | Delete Line | 147 | \|!\|S |
| f8 | Insert Character | 148 | \|!\|T |
| f9 | Delete Character | 149 | \|!\|U |
| Break (f10) | Soft reset | 150 | \|!\|V |
| Copy (f11) | Copy Marked Block | 151 | \|!\|W |
| Cursor Left (f12) | Left one character | 152 | \|!\|X |
| Right (f13) | Right one character | 153 | \|!\|Y |
| Down (f14) | Down one line | 154 | \|!\|Z |
| Up (f15) | Up one line | 155 | \|!\|[ |
| Shift + | | | |
| f0 | Move Block | 156 | \|!\|\ |
| f1 | Swap Case | 157 | \|!\|] |
| f2 | Release Margins | 158 | \|!\|^ |
| f3 | Delete up to Character | 159 | \|!\|_ |
| f4 | Highlight 1 | 160 | \|!space |
| f5 | Highlight 2 | 161 | \|!! |
| f6 | Go to Marker | 162 | \|!" |
| f7 | Set Marker | 163 | \|!# |
| f8 | Edit command | 164 | \|!$ |
| f9 | Delete command | 165 | \|!% |
| Break (f10) | Reboot | 166 | \|!& |
| Copy (f11) | Copy current ruler | 167 | \|!' |
| Cursor Left (f12) | Left one word | 168 | \|!( |
| Right (f13) | Right one word | 169 | \|!) |
| Down (f14) | Down 24 lines | 170 | \|!* |
| Up (f15) | Up 24 lines | 171 | \|!+ |
| Ctrl + | | | |
| f0 | Delete Block | 172 | \|!, |
| f1 | Next Match | 173 | \|!- |
| f2 | Format mode | 174 | \|!. |
| f3 | Justify mode | 175 | \|!/ |
| f4 | Insert mode | 176 | \|!0 |
| f5 | Default Ruler | 177 | \|!1 |
| f6 | Split Line | 178 | \|!2 |
| f7 | Concatenate Line | 179 | \|!3 |
| f8 | Mark as Ruler | 180 | \|!4 |
| f9 | None | 181 | \|!5 |
| BREAK (f10) | Hard reset | 182 | \|!6 |
| COPY (f11) | None | 183 | \|!7 |
| Cursor Left (f12) | None | 184 | \|!8 |
| Right (f13) | None | 185 | \|!9 |
| Down (f14) | None | 186 | \|!: |
| Up (f15) | None | 187 | \|!; |

The possibilities for using function keys in this way are almost limitless. If you can achieve a function by pressing keys, then you should be able to program one of the function keys to carry it out at a single key press - even if this does mean, for more complex functions, the calling up of an EXEC file.

Notes:

1. Ctrl-Shift function keys have no meaning within View until you program them. They generate codes 192 - 216, but the corresponding *KEY Codes (|!@ for f0, and |!A to |!O for f1 to f15) do not appear to work. They can, however, be called up using *FX138,0,n and will operate providing View is in command mode - anything further in the key definition will be ignored as with all * commands.

2. To switch to text mode from command mode use *FX125 or *FX153,0,27, both equivalent to pressing Escape. Anything following either * command will again be ignored. Instead, the command SEARCH/ / can be used to get into text mode without losing any further input; it will position the cursor at the first space.

3. To switch to command mode from text mode use either |[ (i.e. 27), or |!|H (i.e. 136).

4. Other useful codes are:

| | | |
|---|---|---|
| TAB | 9 | \|I |
| Return | 13 | \|M |
| Delete | 138 | \|!\|J |

(127 |? will only work if preceded by a space)

5. All the *KEY codes shown use the 'bar' symbol (|) to be found immediately to the left of the Cursor Left key on the keyboard.

FUNCTION KEY DEFINITIONS 1
(normally used in command mode)

| Definition | Purpose |
|---|---|
| *KEY 0 \|[*L.:0.$.KEYDEF2\|M | Load alternative definitions |
| *KEY 1 \|[LOAD | View LOAD command |
| *KEY 2 \|[READ | View READ command |
| *KEY 3 \|[SAVE | View SAVE command |
| *KEY 4 \|[SAVE :0.V.SAFETY\|M *DIR\|M*SPELL\|M | Ensure file is saved then call SPELLCHECK |
| *KEY 5 | <not defined> |
| *KEY 6 \|[PRINTER | Load printer driver |
| *KEY 7 \|[SCREEN\|M | View SCREEN command |
| *KEY 8 \|[SHEETS\|M | View SHEETS command |
| *KEY 9 \|[PRINT\|M | View PRINT command |
| *KEY 10 OLD\|M | View OLD command |

Notes:

1. Keys 11 - 15 can be programmed and enabled, for command mode, by using *FX4,2. However View disables this feature every time it goes into text mode. Keys 11 - 15 can however be programmed as normal for text mode - see Key Definitions 2.

2 . |[ is included at the start of each definition to ensure that View is in command mode.

FUNCTION KEY DEFINITIONS 2
(normally used in text mode)

| Definition | Purpose |
|---|---|
| *KEY 0 \|[*L.:0.$.KEYDEF1\|M | Load alternative definitions |
| *KEY 1 \|[SEARCH | View SEARCH command |
| *KEY 2 \|[CHANGE | View CHANGE command |
| *KEY 3 \|[REPLACE | View REPLACE command |
| *KEY 4 \|[FORMAT\|M | View FORMAT command |
| *KEY 5 \|[CLEAR\|M*FX125\|M | Clear Markers and return to text mode |
| *KEY 6 \|!\|R\|!'\|!\|R\|!'\|M\|!\|R | Insert extra ruler |
| *KEY 7 \|!#1\|!\|Y\|!#2\|!\|Y\|!\|\ | Transpose letters |
| *KEY 8 \|!#1\|!)\|!#2\|!)\|!\|\ | Transpose word |
| *KEY 9 \|!\|_ | Delete word (Note that a space is needed at end of this line) |
| *KEY 10 OLD\|M | View OLD command |
| *KEY 11 | |
| *KEY 12 \|!\|R\|!$HT\|M1 128\|M \|!\|R\|!$HT\|M2 129\|M | Set default Highlight Codes |
| *KEY 13 \|!\|R\|!$HT\|M1 134\|M \|!\|R\|!$HT\|M2 135\|M | Set Highlight Codes 134, 135 |
| *KEY 14 \|!#1\|[READ:0.V.PASTE 1 \|M*FX138,0,197\|M | Set marker 1, go to command mode Read PASTE file in at Marker 1 CLEAR Markers (= Ctrl-Shift f5) |
| *KEY 15 \|[WRITE :0.V.PASTE 1 2 \|M*FX125\|M | Write Block between Markers 1 and 2 to PASTE file. |

Notes:

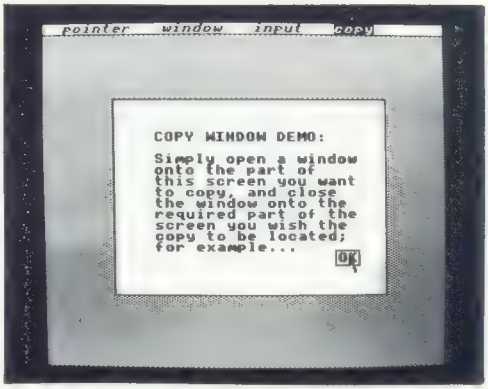
1. The Copy Key cannot be programmed, in either mode.

2. The file V.PASTE has been used here to provide a Cut-and-Paste facility.

# WINDOWS AND ICONS

## (Part 2)

**Alex Kang fills in the detail of his windows and icons routines with a further example of their application.**




If you read the introduction to Windows and Icons last month then you will want to know how to incorporate the USI routines in your own programs. In this month's instalment we deal speedily with all the necessary technical information. A further option has also been added to the demo program from last month showing how the USI routines readily allow any area of the screen to be copied from one place to another. Next month, armed with all this knowledge, we shall describe how to build a complete application program making full use of windows and icons.

Although the USI routines are all in machine code don't worry. They are quite easy to understand and to use within Basic. The technical details are explained below, but remember that most of these routines can be incorporated into easy-to-use Basic procedures as we showed last month.

UNDERSTANDING THE USI ROUTINES

Each of the USI routines is accessed using CALL <label name>, sometimes followed by one or more parameters. These parameters are passed to the routine by assigning their values to the specified integer variables before calling the routine as shown below. You can see the technique in the procedures of last month's demo program.

CALL open,A%,B%,C%,D%,W%

This opens a window on the screen. The text window co-ordinates are held in A%, B%, C% and D% in the same order as in the VDU28 command. The value of W% determines the style of window drawn:

W%=1 gives a shadow window frame
W%=2 gives a double outline frame
W%=3 gives a single outline frame

No frame is drawn for other values of W%. The size of window which may be opened is limited by the space set aside in memory. The total area must not exceed 352 characters. Furthermore, the window must not be more than 32 characters wide.

A text window is automatically defined within the drawn window, using the equivalent of:

VDU28,A%+1,B%-1,C%-1,D%+1

CALL close,A%,B%,C%,D%

This closes a window previously opened using 'CALL open... ' and with the same parameters. The text window is reset to its default dimensions.

CALL pointer

No parameters. This enables the arrow pointer, which is controlled using the cursor keys. Pressing the space bar (making a selection) ends the routine with the position of the pointer stored in &70 and &71.

Before this routine is called, the following values must be set up:

&70 initial pointer x co-ordinate
&71 initial pointer y co-ordinate
T% pointer sensitivity

I normally set them to 20, 16 and 70 respectively. Locations &70 and &71 must not be corrupted while the pointer is in use as these contain the current pointer position. T% can range from 0 to 255: the higher the value, the slower the movement.

CALL prptr and CALL delptr

No parameters. These display or remove the pointer at the current x and y

co-ordinates (held in &70 and &71).

CALL input
No parameters. This accepts input of a number of characters from the keyboard. The maximum number is determined by the previously set contents of N%, but must not exceed 50. The characters are stored in $str, so the label 'str' must be set up before hand (see line 120 in last month's demo program). When using this routine the Escape key must be disabled by *FX200,1.

CALL curon and CALL curoff
No parameters. These routines turn the input cursor on and off respectively.

CALL qubox,x%,y%
This displays a given option with a 'query box' around it. The option string is specified in $str and may be up to 50 characters long. x% and y% must contain the text co-ordinates required for the start of the query box.

CALL oscli
No parameters. This simulates the OSCLI command for compatibility with Basic I. The command string is specified in $str and mustn't be over 50 characters long.

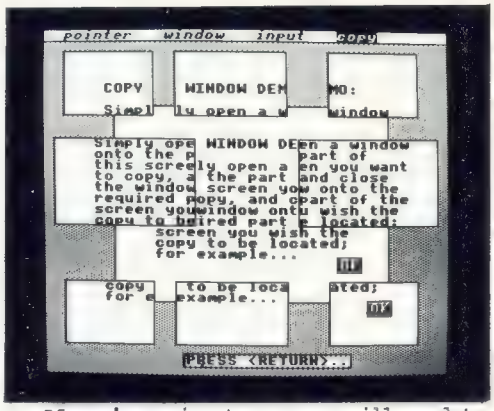
CALL italic
No parameters. This produces italic characters on the screen. The call is best made from within a procedure such as PROCital in the demo program.

Although all these routines may be called directly, most users will probably prefer to use them packaged up in Basic procedures as in the demo program listed last month. The one exception to this is the simple CALL pointer which is best called direct.

USING THE USI ROUTINES

When you run the USI utilities program published last month, a number of labels are displayed on the screen, together with their addresses. These addresses show where the different machine code routines are located in memory. This machine code takes only 1.25K and is relocatable.

If you have a disc machine, you can leave the listing as it is (with PAGE at &1900) and the machine code will lie between &1300 and &1800.




If you're using tape, you will need to relocate the code to &E00 and set PAGE to &1300 (type PAGE=&1300 <Return>). All you need do to relocate the code is to change the value of start% in line 1100 of the USI listing, before running the program. You must also note down the new addresses displayed by the USI program and set them up in your own application program. In last month's demo listing, these addresses were initialised in line 1010. BEEBUG members with a Master 128 will also need to adopt this method as PAGE is set to &E00 even when discs are in use.

The USI routines also need memory to store areas of the screen when displaying 'pop-up' menus. To provide this space, HIMEM is set to &4D00 in line 110 of the demo (listed last month). You must include a similar line in any program of your own.

SOME TECHNICAL NOTES

If you want to change the keys which control the pointer, you may do so by amending lines 4110, 4140, 4180 and 4220 in the USI listing. Look for the LDX command in each of these lines, and replace the following number with the negative INKEY value of the key you wish to use. The space bar may be replaced as the selection key by altering a similar value in line 4310.

If your application program is running short of space, you can claim back part of the buffer used for the pop-up windows. This is normally set at &4D00 (by moving HIMEM), but may be increased, as long as your programs use smaller windows. For each page of memory (256 bytes) you reclaim (and a page is the minimum), the

maximum window size will be reduced by 32 characters.

To alter the start address of the window buffer you have to change line 2280 of the USI listing, which looks like this:

```
2280 LDA #&4D:STA sbf+1:LDX texpar
```

Replace the value &4D, which is the high byte of the current HIMEM address (&4D00), with your required value. You will need to set HIMEM to the same new value in your application program. For example, if you wish to reclaim 1K from the window buffer (reducing the maximum window area by 128 characters), you would change the byte value in line 2280 to &51, and set HIMEM to &5100 in your application program.

## ADDING THE COPY ROUTINE

The following instructions, added to last month's demo program, show how sections of a screen display may be copied from one part of the screen to another. This technique can be used as the basis for a number of different effects and could be combined with other routines to rotate or mirror the areas being copied. The copy feature is selected by choosing 'copy' from the top-line menu display.

You will need to amend four lines of last month's listing to read as follows:

```
 120 DIM str 50:H%=5
 220 IF S%=5 VDU26,12:MODE7:END
 250 DATA 2,8,11,16,19,23,26,29,32,35
2100 ?&D0=2:FORj%=1TO31:PRINTTAB(0,j%)S
TRING$(40,CHR$254);:NEXT:MOVE0,991:DRAW1
279,991:VDU31,0,2:?&D0=0:PROCital("point
er window input copy exit",2,0):ENDP
ROC
```

and add the following lines:

```
 215 IF S%=4 PROCcopywin:GOTO 140
2200 :
2210 DEFPROCcopywin
2220 VDU28,7,24,33,7,12,25,4,224;224;25
,5,1087;224;25,5,1087;799;25,5,224;799;2
5,5,224;224;28,11,23,32,10
2230 PRINT"COPY WINDOW DEMO:"''"Simply
open a window"'"onto the part of"'"this
screen you want"'"to cop y, and close"'
"the window onto the"'"required part of
the"'"screen you wish the" '"copy to be
located;"'"for example...":PROCok(29,21)
2240 PROCcopy:PROCbgd:ENDPROC
2250 :
2260 DEFPROCcopy
2270 RESTORE2300:FORh%=1TO8:READa%,b%,c
%,d%:PROCopen(a%,b%,c%,d%,3):PRINT"Copy.
.":PROCd(40):PROCclose:READ A%,B%,C%,D%:
PROCclose:p%=A%*32:q%=1023-B%*32-31
2280 r%=C%*32+31:s%=1023-D%*32:MOVEp%,q
%:DRAWr%,q%:DRAWr%,s%:DRAWp%,s%:DRAWp%,q
%:PROCd(30):NEXT:PROCopen(13,31,30,29,2)
:PROCqb(14,30,"PRESS <RETURN>..."):REPEAT
UNTIL INKEY-74:ENDPROC
2290 :
2300 DATA 7,12,15,7,2,7,10,2,15,12,25,7
,13,7,23,2,25,12,33,7,28,7,36,2,7,19,20,
12,1,17,14,10,20,19,33,12,25,17,38,10,7,
24,15,19,2,28,10,23,15,24,25,19,13,28,23,
23,25,24,33,19,28,28,36,23
```

The copy routine uses the fact that when a window is opened, its background information, stored in the window buffer (AT &4D00 TO &5800), will remain until another CALL open command. Thus closing the window onto another part of the screen (by altering the parameters A% to D%) simply outputs the present buffer contents onto that part of the screen. To create multiple copies, just keep closing the window onto the required parts of the screen, so long as no other window is opened. In the demonstration this is used to 'tear apart' a screen display.

The complete demo program (and all the USI utilities for those who missed them last month) are included on the magazine cassette/disc. We recommend that you study the demo program carefully and use this as the basis for your own experimenting, in the first instance, before you start to design and write your own programs using the USI routines.

## FURTHER IDEAS

I am sure that some of you will have better ideas on how to make use of these utilities. One possibility (for disc users) is to have multiple overlapping windows, by *SAVEing and *LOADing the window buffer from &4D00 to &5800. Just be careful that when you close each window, the parameters are restored to the same values as when the window was opened.

The final article on windows and icons, in the next issue, will describe a complete and practical application of the USI routines described here.

# THE MORLEY TELETEXT ADAPTOR

**Acorn's Teletext adaptor for the BBC micro has proved less popular than hoped. Geoff Bains reports on a cheaper alternative from Morley Electronics that may be set to change all that.**

**Product : Teletext Adapter**
**Supplier : Morley Electronics,**
**1, Morley Place, Shiremoor,**
**Tyneside, NE27 0QS.**
**091-262 7507**
**Price : £74.45 (adapter)**
**£24.95 (ROM software)**
**£23.95 (disc software)**
**£9.95 (power supply)**

When the BBC micro was first unveiled one of its many expansion capabilities was the facility to receive and display Teletext information directly from a TV aerial and to download software via the Teletext service. Within three years or so Acorn finally produced its Teletext adaptor for a price of £225.

Acorn Teletext adaptors have not sold in vast numbers even though the price has since dropped to around £140. The field has been open for an enterprising company to produce a Teletext adaptor for a more reasonable price. This is the position in which Morley Electronics hopes to find itself. The Morley Teletext adaptor can cost as little as £75, though a more reasonable system (with driving software) is around the £100 mark.

Morley's adaptor comes in a plastic half-height disc drive case - itself an advantage over the second processor sized box of Acorn's - which plugs into the User Port and the auxiliary power supply under the Beeb, and a TV aerial. If your disc drives already use the auxiliary power socket you will need Morley's power supply too, for £10. Why Morley chose to use the User Port is anyone's guess. There is already a section of the 1 MHz bus memory map set aside for Teletext so using this, and leaving the User Port free for mice and the like, would have been preferable.

If you choose to use Morley's software (and it would be silly to buy the adaptor without it) you have a choice of either ROM based software or a disc-based program for sideways RAM. Either way, the effect is much the same. It will allow you to display Teletext pages from any TV channel, download software, save broadcast screens to disc or cassette, and even to tell the time; all without consuming vast quantities of RAM and raising PAGE, as does Acorn's Teletext ROM.

The first job to do when you unpack your new adaptor is to tune it to the frequencies of the four TV stations in your area. This is all done electronically and automatically - a great improvement over the twiddling around the back with a screwdriver necessary with Acorn's adaptor. Once the adaptor has found the four stations, it saves the tuning


information to disc or cassette for next time. You can even blow your own EPROM (or get Morley to do it for you) to have this information always to hand whenever you use the adaptor.

Now you can actually receive Teletext information. A simple menu screen allows you to select the channel, page number, and so on required and the chosen page is then displayed. Similarly you can select the downloading feature from the menu and transfer the free software broadcast with Teletext into your machine ready for running or saving.

Once a page is up on the screen you can just read it, hold it there, reveal any concealed parts, step on to linked pages with the cursor keys, or store it away on cassette or disc.

Morley's Teletext software does not end there. There are over 90 * commands available for you to make full use of the adaptor; all of which use the Beebug convention of an optional 'M' prefix to avoid command clashes. These range from convenience commands, such as *ITV2 to tune to Channel 4 and *KIDS to display the jokes pages on BBC 2, to building block commands for your own control programs, such as *TRANSFER to store a page in memory and *ROW to read a specified page row into memory, and several dozen more. There's also provision for printing the date and time as these are broadcast along with the Teletext data.

With such a plethora of commands you would be forgiven for expecting a pretty


hefty manual to accompany the Morley Teletext adaptor. However, the tome that does come with the package is anything but comprehensive. It is true that a good setting-up and fault-finding section is included and each command is listed along with a (very) brief description of what it does but there is very little in-depth explanation as to what use half of these commands could be put. More importantly, even the simple introduction to using the adaptor in its simplest form - to display Teletext pages and to download software - is brief to the point of downright misleading. You really have to experiment for yourself with this package before you can get to grips with it.

All this is a great shame as the adaptor itself, although not built to the same standard, is better than Acorn's in many respects, and cheaper, and the software could not be more comprehensive.

WHAT IS TELETEXT?

Teletext is a free service run by the television broadcasting authorities that utilizes sections of the broadcast TV signal to transmit digital data in a standard format. Teletext data may be received by any suitable receiver and decoder connected to a normal TV aerial without any payment on the part of the receiver except for a TV licence.

Most Teletext receiver/decoders are built into TV sets (so-called 'Teletext TVs'). However, the Teletext-compatible mode 7 display of the BBC micro means that much of a Teletext receiver/decoder hardware already exists in this machine. A Teletext adaptor turns the Beeb into a Teletext-receiving terminal capable of all the functions of a Teletext TV except the superimposition of subtitles over broadcast programmes.

The Teletext information is broadcast as a series of 'pages', each being a mode 7 screenful. The information covers a wide range of subjects - travel news, share prices, weather, TV programs, recipes, jokes, etc. - and also software for the Beeb. This software similarly covers a reasonable range of subjects, though with a heavy schools bias, and is only available to owners of Teletext adaptors. Normal Teletext TVs are not able to download software.

# 6522 Timers (Part 2)

**This month, Surac takes a further look at the Beeb's internal timers and gets involved in interrupts and events.**

Last month, I looked at the 6522 Versatile Interface Adaptor (VIA) used in the Beeb. In particular, I described the timers in the chip and explained how they could time events with an accuracy of 1 micro-second.

This month, we'll go on from there to use Timer 1 (T1) in the VIA, in conjunction with the computer's system of interrupts, to set up a 1 ms clock alongside the built-in 10 ms timer.

## INTERRUPT HANDLING

An interrupt is a signal to the computer. When it occurs, it makes the micro stop what it was doing and go off to do whatever the interrupt demands ('servicing' the interrupt). Having done this, it carries on where it left off.

The way that the Beeb handles raw interrupts is briefly covered on page 466 of the User Guide, which tells us that the computer has a pair of interrupt vectors, IRQ1V and IRQ2V. These vectors hold the starting addresses of the machine code service routines and represent the computer's priorities.

When an interrupt occurs, the OS jumps to the IRQ1V code. This identifies and handles all the normal system interrupts. If it cannot recognise an interrupt's source, the computer will jump to the address in IRQ2V, where it assumes will be code to handle the 'alien' interrupt. Control then returns to the original program.

Either vector can be used. However, IRQ1V is meant for high priority items, so stick with IRQ2V (at address &206/&207). This must contain the address of the start of our code. The new routine can end in one of two ways. The easy way is to use an RTI (ReTurn from Interrupt) instruction. However, better is to save the contents of the original vector and to leave the routine with an indirect jump to this address, preserving the interrupt 'chain':

```
JMP (oldvector)
```

After the service routine is completed the machine must be returned to its pre-interrupt state so, at the start of the routine, save the processor registers and location &FC (which holds the accumulator contents on entrance to the routine) onto the stack and pull them off at the end. Make sure that the interrupt handler does not run for too long; anything much over 1 ms could affect the system.

## PRODUCING TIMER INTERRUPTS

Interrupts on the 6522 must first be 'enabled' via the chip's 8-bit Interrupt Enable Register. This is at address &FE6E on the Beeb and it controls all 7 sources of VIA interrupts. A 1 in any bit position enables the associated interrupt, while 0 disables it. Bit 6 of the IER controls T1 interrupts. To set bits in the IER, write data to it with the MSB and the required bit set to 1. To clear bits, write with MSB=0 and 1 in the required bit positions.

Whenever the VIA generates an interrupt, a bit is set in the 6522's Interrupt Flag Register (IFR - at address &FE6D) to identify its source. Bit 6 indicates that a T1 interrupt occurred.

## AN EXTRA TIMER

With last month's article, we've now got enough information to set up an extra timer which can be used, from Basic, to time things to within 1 ms:

```
10000 DEF PROCassemble
10010 REM ** Set up system constants
10020 IRQ2V=&206
10030 t1cl=&FE64:t1ch=&FE65
10040 acr=&FE6B:ifr=&FE6D:ier=&FE6E
10050 period=1000:REM** Microseconds
10060 FOR PASS=0 TO 3 STEP 3
10070 P%=&A00:REM** &D10 for cassette
10080 timbuff=P%:oldirq2v=P%+4:P%=P%+6
10090 [OPT PASS
10100 \
10110 \Initialise interrupt system
10120 .init
10130 SEI \Stop interrupts
10140 LDA IRQ2V:STA oldirq2v
10150 LDA IRQ2V+1:STA oldirq2v+1
10160 LDA #(inthandle MOD 256):
      STA IRQ2V
10170 LDA #(inthandle DIV 256):
      STA IRQ2V+1
10180 LDA #(period MOD 256):STA t1cl
10190 LDA #(period DIV 256):STA t1ch
10200 LDA #&C0:STA acr \Start T1
10210 LDA #&C0:STA ier \Enable T1 ints
10220 LDA #0:STA timbuff:STA timbuff+1
10230 STA timbuff+2:STA timbuff+3
10240 CLI \Re-start interrupts
10250 RTS
10260 \
10270 \Shut down interrupts
10280 .finish
10290 SEI
10300 LDA oldirq2v:STA IRQ2V
10310 LDA oldirq2v+1:STA IRQ2V+1
10320 LDA #&40:STA ier \Disable ints
10330 LDA #0:STA acr \Stop timer
10340 CLI
10350 RTS
10360 \
10370 \Handle the interrupt
10380 .inthandle
10390 PHA:PHP:LDA &FC:PHA
10400 LDA ifr  \Interrupt from..
10410 CMP #&C0 \..Timer 1?..
10420 BNE done \..if not, exit
10430 .timint
10440 LDA t1cl \Clear int. flag
10450 \Bump timer registers
10460 INC timbuff:BNE done
10470 INC timbuff+1:BNE done
10480 INC timbuff+2:BNE done
10490 INC timbuff+3
10500 .done
10510 PLA:STA &FC:PLP:PLA
10520 JMP (oldirq2v) \To orig. handler
10530 ]:NEXT:ENDPROC
```

The code is not complex and has three main parts. 'Init' sets up the interrupt system and points the IRQ2V vector to the new interrupt handler, saving the original pointer. Timer 1 is set to free run, giving an interrupt every 'period' microseconds (set to 1000). Each time that it reaches zero and interrupts, the registers are reloaded with the value of 'period' stored in the latches to give a steady series of pulses. Finally, it zeros the new clock counter and enables the interrupts. Note how interrupts are disabled during 'init' by the SEI command to prevent any nasties in the middle of changing the vector.

'Finish' disables the T1 interrupts and resets IRQ2V to its original value. Again, interrupts are disabled while the vector is being changed. The main handler routine, 'inthandler', first checks to see if the interrupt has come from the timer. If not, then the routine leaves by jumping to the original value of IRQ2V. If a T1 interrupt did occur, the program clears it by reading T1CL. It then increments the 32-bit counter at 'timbuff' which is our extra clock. The system ends by jumping to the original IRQ2 handler to see if there is any other work to do. Note that the values of the accumulator and program counter registers, the only ones used, are saved in the stack through the handler.

To use the routines from Basic, first assemble them. They can then be started at any time with:

```
CALL init
```

This zeros and starts the clock. You can read and write the clock with:

```
<lineno> value=!timebuff
<lineno> !timebuff=value
```

You will find that the clock noticeably slows down other functions on the micro. The effect is most obvious when writing to the screen - try listing the program with the clock going. This slowing can be reduced by increasing the timer period to, say, 10 ms (change 'period' to 10000 in line 10050) though this will, of course, lose some accuracy in the added clock. It is best to switch the timer off whenever you don't need it with:

```
CALL finish
```

The timer is very useful, particularly in, say, a school or lab when the micro is being used to monitor and/or control outside events. This month's magazine cassette/disc has a demo program on it to show the timer in use.

# INTERACTIVE 3D

**Although graphics packages have proved popular for the BBC micro, good ones can be hard to find. Terry Hallard reports on one example that deserves greater recognition.**

**Product : Interactive 3D**
**Supplier : Design Dynamics**
**8 Meadow Way, Ampthill,**
**Beds, MK45 2QX.**
**Price : £12.95 (disc)**
**£8.95 (cass.)**

The creation, manipulation and display of three dimensional images is an important part of the armoury of any respectable industrial computer. The images generated on these mainframes and minis can, because of the speed and size of such machines, be of solid or wireframe form and each type of representation has its own strengths and weaknesses.

One of the first of these programs for a micro was Psion's VU-3D for the Spectrum and this was the only thing for which I have ever cast envious eyes Sinclairwards. But no longer, as there are now several packages which will perform the same function on the Beeb. Notable among these are AMX's 3D-Zicon (reviewed last month) and Interactive 3D - the subject of this review.

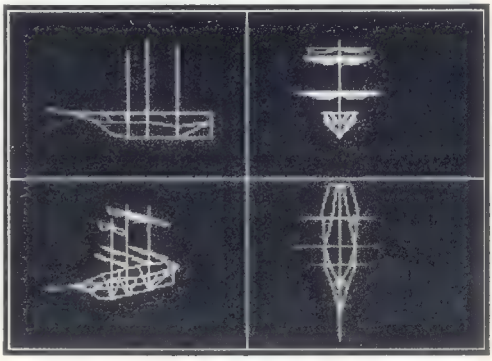
The images generated by these, as with all other microcomputer 3D packages, are limited to wireframe models because of memory limitations. However, such representations are reasonably satisfactory to look at in most circumstances. Problems do arise sometimes when your brain cannot decide, because of the ambiguous evidence of the lines on screen, exactly which orientation the object is in - an optical illusion rather like those cube pictures which 'flip' between standing on one face or another.

Interactive 3D is essentially keyboard only, with lots of dedicated keys to learn. The package relies on the function keys on their own and with Shift and Ctrl, giving thirty possible commands on the keystrip supplied. These are explained in the seven page guide that accompanies the program.

Clever use is made of the cursor keys to hop around the flat two dimensional drawing to any desired point. Pressing f0 (Find point) 'acquires' that point three dimensionally, presenting its Cartesian XYZ co-ordinates in a two line message window above the workbox. Pressing f2 creates a new point superimposed on the fixed one. This new point is then moved, with the appropriate directional keys, to the desired location, keeping at all times a straight line between it and the old point. If you do happen to wander a little off course, then a press of Shift-f1 will align the line with any co-ordinate that the previous line is in. Points or lines can also be deleted from the picture with a single key press.

The principles of 3D manipulation take a little getting used to and necessitate six dedicated directional keys to control bi-directional movement along each of the three axes. To aid the user in picking the correct direction, Interactive 3D has, at the centre point, a useful cluster of X, Y and Z pointers. These pointers can be shown or deleted as needed. It becomes quite easy to dispense with them after a while as they do tend to obliterate the centre of a fiddly drawing like the ship shown here.

The major thing that I like about

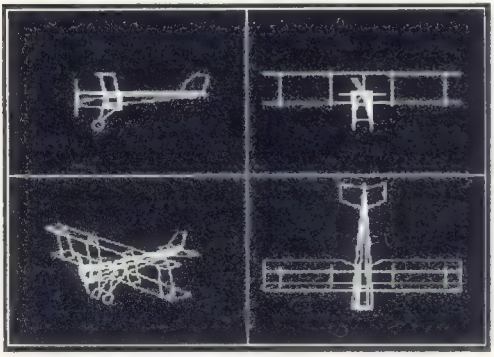


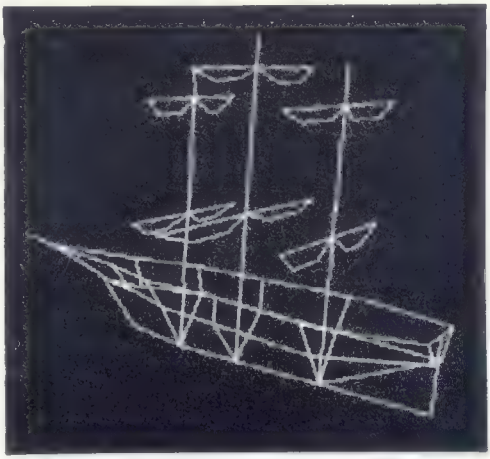
Interactive 3D is the choice of view that the user can work in. Single key presses allow the user to work on the front, side or plan view or with a perspective view which can be rotated to any attitude one wishes. But the main display is, as shown, a quartered screen which shows three orthographic views in First Angle Projection and a perspective view all at the same time. The user draws with points on the perspective and can see the results appear on the other views. This makes it much easier to move your cursor in the 3D direction that you really intend.

The worst part of creating 3D images is the sheer tedium of plotting each of many points in three-dimensional space - which is, of course, represented by only a two-dimensional drawing on the screen. Each package has its own solutions to these two problems.

Interactive 3D requires only half of a symmetrical object to be defined - the rest is created by reflection about a defined axis. Judicious use of this process at various stages of a complex drawing can cut the workload by much more than half. By 'acquiring' two points, a line can be drawn between them with a single key press. The program is capable of supporting 420 points and 630 lines, which can make for a very complicated drawing indeed.

The program has several other very useful features. The centre of rotation can be changed to any point on the model with the result that the model is shifted to one side or other on the screen which is a very crafty way to 'pan' around the





picture. There is a zoom facility that allows the model to be blown up or shrunk to requirements and, of course, the perspective model can be rotated to a new angle (though the rotation is via the cursor controls so that the rotation is rather hit and miss and I have yet to really get the hang of this). In both zoom and rotation the Shift and Control keys along with another key give, respectively, coarse or fine control

After either of these transformations has been carried out a press of Shift-f7 aligns the whole model with the new axes arrangement.

Drawings can be saved and reloaded with a disc or cassette system; the foreground and background colours may be changed to the individual user's preference, as can the viewpoint of the perspective display. When your masterpiece is finished the picture can be dumped to an FX80 or compatible printer. I found that it would not dump to my RX80 but a suitable dump ROM overcomes this problem. Normally only the perspective view is printed, even after Escape is used. To get the whole screen I had to type in VDU26 to reset the screen window.

Despite these little quirks, and after an initial period of hard and sometimes frustrating practice, I enjoyed using this program very much. I feel that it has got a lot to recommend it - not the least being the price.

# Across the Tube
## (Part 1)

**Much of the workings of the Tube have never been revealed by Acorn. Ian Trackman, a regular contributor to BBC Television and author of several software packages and books, lifts the veil.**

THE TUBE

The BBC micro is unique amongst home micros in that it was designed from the outset to be coupled with another computer to increase its power. The second computer is not really what we would call a 'computer'. It has no keyboard nor display and so it is just called a 'Second Processor'. Instead of communicating with the outside world the Second Processor communicates with the original Beeb (now the 'Host' Processor). The Beeb becomes its ears, eyes, voice, and hands, looking after the tasks associated with the outside world - the screen, sound, file storage, and so on. The Second Processor is left with the time to process the program that you want to run much faster.

There is a variety of Second Processors available for the Beeb and the Master, designed for a variety of purposes. For this article we are concerned only with the 65C02 Second Processor - what you might call the general purpose Second Processor.

When a Second Processor is fitted to a BBC micro the two remain separate entities, linked only by a hardware/software umbilical cord called the 'Tube'. At the simplest level (Basic) you need not worry that this exists at all or that your system is in fact two machines in one. The Basic interpreter looks after it all. However, if you have a desire to program the two-processor system in machine code you will find a severe lack of information on the workings of the Tube. This article attempts to correct this.

THE NEED TO COMMUNICATE

To get the most out of your two processor system, both processors should be continually busy. If one is left waiting for the other to finish one task before it can get on to the next then the extra hardware is wasted. Effective communication between the two is essential for effective operation.

The normal operation of the two-processor system is with a master program running in the Second Processor, using one or more 'slave' programs in the Host Processor and controlling the whole system. Typically, the master program will use OS facilities, (e.g. OSBYTE calls) and filing system calls, both of which will be handed down to the Host Processor under the control of the Tube software. However, you may wish to supplement those facilities with your own routines, running in the Host Processor. The reasons for this may be:

1. Lack of space in the Second Processor.

2. To protect the routines from corruption by activities in the Second Processor.

3. To inform the Second Processor of a change in the system's environment (e.g. the occurrence of a particular interrupt in the Host Processor).

4. To improve operating response times (to keep both processors running flat out as mentioned earlier).

The last two reasons are the most probable, and the ones on which this article will concentrate. They require a number of facilities of our trans-Tube communication:

1. Data read by Second Processor from the Host Processor.
   Data written by Second Processor to the Host Processor.

2. Data read by Host Processor from the Second Processor.
   Data written by Host Processor to the Second Processor.

3. Routine in Host Processor called by Second Processor.
   Routine in Second Processor called by Host Processor.

4. Host Processor interrupts handled by the Second Processor.

OSWORD CALLS 5 AND 6

The official method of sending data across the Tube is to use OSWORD calls 5 and 6. (See the Advanced User Guide page 249). These enable a program in the Second Processor to, respectively, read and write data across the Tube to the Host Processor's RAM. The call is not very fast and, because it uses the A, X and Y registers, a block transfer can be over 50 times as slow as a simple 'indexed' loop within one processor, such as:

```
.loop
LDA (source),Y
STA (destination),Y
INY
BNE loop
```

When using these OSWORD calls, the X and Y registers should point at a parameter block containing the four byte address of the byte of data. The byte read (in the case of OSWORD 5) is returned in the fifth byte of the block. Here are two example subroutines for use with block transfers. It is assumed that the parameter block is already set up at 'ioblock'.

```
.ioread
LDA #5
JSR iosub
LDA ioblock+4
RTS

.iowrite
STA ioblock+4
LDA #6
JSR iosub
RTS

.iosub
STX xsave          \ Preserve the X and Y
                   \ registers
                   \ (or as required)
STY ysave
LDX #ioblock MOD &100
LDY #ioblock DIV &100
JSR osword
INC ioblock        \ Ready for next call
BNE iosub2         \ Unless over page
                   \ boundary
INC ioblock+1
.iosub2
LDX xsave          \ Restore X & Y values
LDY ysave
RTS
```

The advantage of the OSWORD calls is that the transfer process is transparent to the calling program. However, their disadvantage is that only one byte of data can be transferred at a time.

OTHER METHODS OF COMMUNICATION

Another possible, but 'unofficial', route for data from the Second Processor to Host Processor is the use of commonly used OS calls, such as OSBYTE, OSWORD, OSWRCH and OSRDCH. These pass through vectors in the Host Processor on their way to execution by the OS. Data to be passed across the Tube can 'hitch a lift' with these official travellers as parameters.

Most of the OSBYTE and all of the OSWORD calls take parameters in both X and Y registers. Issuing the appropriate call from the Second Processor, enables two bytes of data to be transferred to the Host Processor as parameters in these registers. The method is to intercept the call as it passes through the vectors in the Host Processor and to trap the parameters in a user routine. (Remember, however, that the Y register is fetched from the Host Processor only in the case of OSBYTE calls with A greater than or equal to &80.

However, some OS facilities are handled entirely within the Host Processor, so they cannot be accessed directly from the Second Processor. Consequently, it is not possible to issue such calls from the Second Processor. Either the Tube software bypasses the Second Processor vectors completely (e.g. on a *CODE call) or the situation giving rise to the routine occurs only in the Host Processor (e.g. counting the contents of a buffer). Attempting to issue any of these routines (with an indirect jump to the vector address in the Second Processor) results in a 'Bad' message.

These unserviced vectors are:

| | | |
|---|---|---|
| USRV | VDUV | IRQ2V |
| KEYV | FSCV | UPTV |
| INSV | REMV | NETV |
| | CNPV | |

However, trans-Tube communication is still possible through some of these calls. We will look particularly at the User Vector and the VDU Extension Vector, although the principles can be applied to most of the others.

THE USER VECTOR

Two sets of calls pass through this vector in the Host Processor. The first set of calls consists of *CODE and *LINE. *LINE does not have an official calling address and can therefore only be used from a machine code program by using the OSCLI routine to recognise the characters of 'LINE'. *CODE can be performed by OSBYTE &88. X and Y are preserved on entry to the vector, enabling them to be used for data passing, but A is cleared to zero.

The second set of calls are the (currently) undefined OSWORD calls &E0 to &FF. As before, the X and Y registers are preserved but, in addition, the A register's value is also unchanged, thus providing the opportunity of passing a further parameter which is 5 bits wide. By masking out the top bits (e.g. AND &1F), a value in the range 0 to &1F can be passed across the Tube.

THE VDU EXTENSION VECTOR

Unrecognized VDU calls are passed through this vector (see the Advanced User Guide page 261). Since VDU commands issued from machine code in the second processor consist of a series of OSWRCH calls each issuing a single byte, there does not appear to be any significant benefit in using this vector, excepting that the parameters are buffered on the Host Processor in the 'VDU variables' buffer, so that it would be possible to send up to 8 variables in a string ready for subsequent collection on the Host Processor. The Host Processor side of the Tube has a ten-byte buffer, but the first character of a VDU list is not buffered and the second character of a VDU 23 call has a limited range.

Note, however, that the proper function of this vector is to support extended graphics ROMs and Acorn does not advise its use for other trans-Tube communications.

FAST SINGLE BYTE TRANSFER

A further possibility for communication is the fast Tube BPUT OSBYTE call &9D. This transfers its X and Y registers across the Tube in the same way as other OSBYTE calls. Its special property is that control is returned to the Second Processor immediately, so providing a noticeably faster response time than an OSWORD 6 call. The drawback is that the call vectors through the standard BYTEV vector in the Host Processor. To trap this call (trap A=&9D) and collect the parameters, therefore, all OSBYTE calls must be re-vectored.

INSTALLING A USER ROUTINE IN THE HOST PROCESSOR

All these trans-tube communication methods require a routine to be present in the Host Processor. Therefore, before we can communicate we need to install the routine in the Host Processor. There are two methods of doing this. The first is to *LOAD the program from tape or disc by designating the re-load address with the top 16 bits set, for example:

```
*LOAD PROGRAM FFFF2800
```

The second method is to load the program into the Second Processor (or assemble it there) and then transfer the bytes of the object code across the Tube with OSWORD 6 (which being the 'official' route requires no user code in the Host Processor). Remember that you are effectively transferring data bytes and so you must take care with internal program references (such as JSRs to other parts of the program) to ensure that the address values are correctly adjusted.

Alternatively, assemble the program with an offset (setting bit 3 of the OPT parameter) or set its origin to its corresponding final Host Processor address. Remember that with the Tube operational, the character font in the Host Processor is expanded, so a disc system's Host Processor free RAM starts at &2100 and extends up to the start of screen memory.

Even the area starting at &2100 is used by some filing system calls. For

example, *COPY and *COMPACT can overwrite large parts of the Host Processor's available RAM. The memory normally reserved for the current language (&400 to &7FF) is also unavailable as it is taken over by the Tube software. Locate your Host processor routine with care!

CALLING THE HOST PROCESSOR ROUTINE

Instead of *LOADing the program, you can also *RUN it. As you will probably wish to access the program again, its first task should be to set up an entry vector to a 'warm start' entry point somewhere else in the program.

Using *RUN has the disadvantage that it is harder to pass parameters to the program. The values of A%, X% and Y% are ignored by *RUN , as the call will not be made by the Basic CALL command. If you want to pass one or more parameters, you will have to append them as string values after the filename and have the program collect them with an OSARGS call (with A=1).

Perhaps the 'cleaner' method is to alter the appropriate vector in the Host Processor (depending whether you have decided to use OSBYTE &9D, *CODE, an unrecognized VDU command or whatever). The two bytes of the vector should be changed (with OSWORD 5) to point to the start of your own routine. Before doing so, I suggest that you preserve the original vectors - either, say, at the end of your program or in one of the spare vectors IND1V, IND2V or IND3V.

It is, of course, important that you don't carry out the re-vectoring operation twice, otherwise the original vector will be lost and both vectors will point to your program! One way of preventing this from happening is to compare the vector value with the entry address of your program. If they are the same, the vector can be presumed to have already have been changed. Another method (which will not work with OS 1.0) is to change the main vector in the normal way, but then initialize the second vector with the first vector's default address as read from the default vector table pointed to by &FFB7 and &FFB8.

COMMUNICATING FROM THE HOST PROCESSOR

There is no 'official' route for a program in the Host Processor to pass data to the Second Processor. However, one approach is to execute (from the Host Processor) an OS call which does not affect the operating environment but which takes one or more parameters. The call could be trapped through its vectors in the Second Processor and the data extracted. For example, OSBYTE 0 (to identify the OS version) returns a value in X. Provided that X<>0 on entry (to inhibit any display), a parameter can be passed in the Y register.

OSWRCH could also be used to pass two parameters in X and Y. The vector trap could identify the 'fake' call if the character to be printed were, say, CHR$0. (But be very careful if you try to use this trick with VDU calls which take multiple parameters e.g. VDU19). However, this method is untidy and can interfere with the fast running of the foreground program.

NEXT MONTH

Next month we conclude our look at communicating across the Tube and see how to deal with interrupts.

## POINTS ARISING POINTS ARISING POINTS ARISING POINTS

THE EARTH FROM SPACE (BEEBUG Vol.4 No.8)

In sqeezing this program listing into the available space a line number was left hanging in mid-air. Inserting the line '70 REM' will overcome this problem.

WIZZARD (BEEBUG Vol.4 No.8 magazine cassette/disc only)

If you clock up 99 minutes when playing this game the program will crash with the message "String too long at line 2430". Inserting the following line into the program will prevent this happening, although the time display will remain at the same value from there on.

```
2420 IF TIME>594000 THEN TIME=594000
```

# JASON MASON

**Jason Mason is a game with a difference. Can you turn a random pile of building blocks into a house, a boat, or a bus? Richard Lewis explains.**

Jason is a hard-working mason. He loves to build different objects, and the quicker he builds them, the sooner he can start on the next.

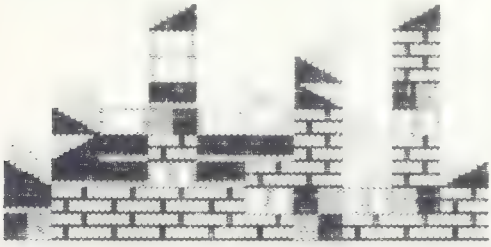
But building complicated objects is not at all easy for Jason, and he needs your help and guidance. Can you, with expert skill and judgement, help Jason to create a masterpiece?

The object of the game is to move a set of building blocks, one by one, from a pile and create a finished object.

There are 10 objects included in the listed program which you can build. These are:

| | |
|---|---|
| Express | Church |
| Liner | Train |
| Tapestry | Bus |
| Boat | House |
| Bungalow | Castle |

The keys used to control Jason are 'Z' and 'X' for left and right, and '>' and '?' for pick up and drop left and right. To pick up a brick from Jason's left and drop it to his right, you should press '>' then '?'.



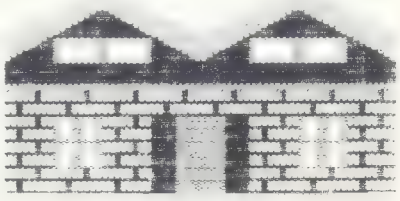
Every time you pick up a building block, your score also decreases. If your score falls below zero, Jason will die from exhaustion.

If you are not sure of the shape of the construction that you are trying to build, then pressing 'D' will reveal the completed object, but this only lasts for a couple of seconds, and takes a fair sized chunk from your score.

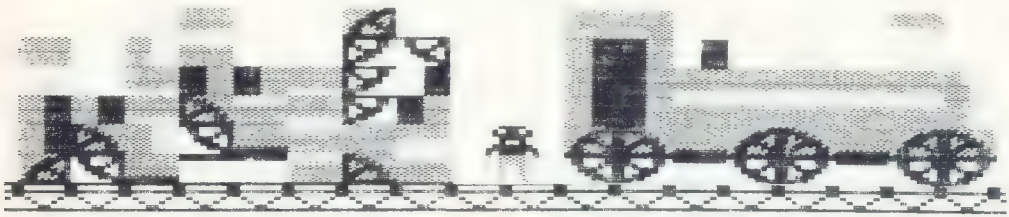
Jason's movement is not restricted by the partly built objects or blocks which may be lying around, but he cannot walk up or down steps of more than one block at a time.

Do not RENUMBER the program as it uses calculated RESTOREs at lines 2160 and 2180.

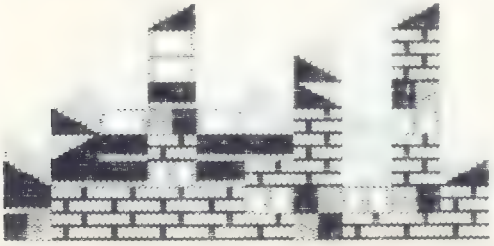
```
  10 REM PROGRAM JASON MASON
  20 REM VERSION B0.2
  30 REM AUTHOR  R. LEWIS
  40 REM BEEBUG  APRIL 86
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 ON ERROR GOTO 200
 110 MODE5:VDU23,1,0;0;0;0;
 120 DIM HSC%(10),A%80,Z%800,R%200
 130 ?&D0=2:VDU23,1,0;0;0;0;
 140 PROCCHARS
 150 PROCINST
 160 PROCINIT
 170 REPEAT:PROCMAN:PROCCHECK:UNTILFIN%
 180 GOTO150
 190 :
 200 ON ERROR OFF
 210 MODE 7:?&D0=0:IF ERR=17 END
 220 REPORT:PRINT " at line ";ERL
 230 END
 240 :
1000 DEF PROCINIT
1010 FOR I%=1TO3
1020 READ V%:VDU19,I%,V%,0,0,0:NEXT
1030 CLS:COLOUR1:FORI%=0TO19
1040 PRINTTAB(I%,31);CHR$227;:PRINTTAB(
I%,16);CHR$228;
```

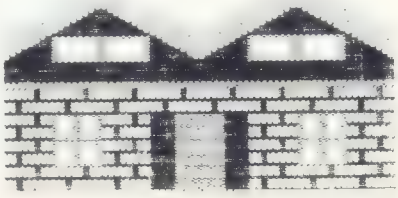


```
1050 K%=I%+1+A%:K%?20=0:?K%=0:NEXT
1060 FORI%=0TO400:Z%?I%=0:NEXT
1070 FORI%=0TO100:R%?I%=0:NEXT
1080 SC%=0:READ X1%:READ X2%:READ V%:FO
R I%=1 TO V%:FOR J%=X1% TO X2%
1090 P%=(I%-1)*10+J%:READ Q%:R%?P%=Q%:I
F Q%=0 GOTO 1140
1100 SC%=SC%+50:BN%=Q% MOD16:C0%=(Q%DIV
16)MOD4:C1%=Q%DIV64
1110 COLOURC0%:COLOUR128+C1%:X%=4+RND(1
0):K%=X%+1+A%:Y%=?K%:IF Y%>8 THEN GOTO 1
110 ELSE ?K%=?K%+1:PRINTTAB(X%,15-Y%);CH
R$(235+BN%);
1120 K%=Y%*40+X%+1+Z%:?K%=Q%
1130 COLOUR1:COLOUR130:PRINTTAB(5+P%MOD
10,31);CHR$226;
1140 NEXT:NEXT
1150 COLOUR128:M%=0:IM%=0:IB%=0:IX%=0:X
%=0:IY%=0:Y%=15:B%=0:RN%=0:PROCMANCH:PRO
CMAKE:E$=E2$:M$=M2$:FIN%=FALSE:OSC%=SC%
1160 COLOUR3:PRINTTAB(5,1)"SCORE ";SC%;
1170 ENDPROC
1180 :
1190 DEFPROCMANCH
1200  ON 2*M%+IM%+1 GOTO 1210,1220,1230
,1240,1250,1260,1210,1220
1210 IF B%=0 THEN VDU23,230,189,189,60,
36,36,36,100,6:GOTO 1270 ELSE VDU23,230,
60,60,60,36,36,36,100,6:GOTO 1270
1220 IF B%=0 THEN VDU23,230,189,189,60,
36,36,36,38,96:GOTO1270 ELSE VDU23,230,6
0,60,60,36,36,36,38,96:GOTO1270
1230  VDU23,230,60,60,24,24,28,22,19,50
:GOTO 1270
1240  VDU23,230,60,60,24,24,56,232,72,2
4:GOTO 1270
1250  VDU23,230,60,60,24,24,56,104,200,
76:GOTO 1270
1260  VDU23,230,60,60,24,24,28,23,18,24
:GOTO 1270
1270 ON M%+B%+1 GOTO 1280,1290,1300,134
0,1310,1320,1330,1340
1280 VDU23,231,60,90,126,60,36,126,255,
189:GOTO 1340
1290 VDU23,231,28,6,62,20,28,60,126,189
:GOTO 1340
1300 VDU23,231,28,48,62,20,28,60,126,18
9:GOTO 1340
1310 VDU23,231,189,219,255,189,165,126,
60,60:GOTO 1340
1320 VDU23,231,157,135,191,149,157,126,
60,60:GOTO 1340
1330 VDU23,231,157,177,191,149,157,126,
60,60:GOTO 1340
1340 ENDPROC
1350 :
1360 DEF PROCCHARS
1370 VDU23,226,255,255,255,0,0,255,255,
255
1380 VDU23,227,255,255,255,255,255,255,
255,255
1390 VDU23,228,255,129,255,129,66,36,24
,255
1400 VDU23,229,255,129,129,129,129,129,
129,255
1410 VDU23,235,1,3,7,15,31,63,127,255
1420 VDU23,236,128,192,224,240,248,252,
254,255
1430 VDU23,237,240,240,240,240,240,240,
240,240
1440 VDU23,238,192,240,184,156,188,230,
198,255
1450 VDU23,239,255,198,230,188,156,184,
240,192
1460 VDU23,240,255,99,103,61,57,29,15,3
1470 VDU23,241,3,15,29,57,61,103,99,255
1480 VDU23,242,0,0,0,0,255,255,255,255
1490 VDU23,243,251,251,251,0,223,223,22
3,0
1500 VDU23,244,255,153,153,153,153,153,
153,255
1510 VDU23,245,0,24,60,126,126,255,255,
255
1520 VDU23,246,24,24,24,255,255,24,24,2
4
1530 VDU23,247,255,129,129,129,129,129,
129,255
1540 VDU23,248,255,255,0,0,0,0,0,0
1550 VDU23,249,0,0,0,0,0,0,0,255,255
1560 VDU23,250,153,102,66,66,102,153,0,
0
1570 ENDPROC
1580 :
1590 DEF PROCMAKE
1600 M2$=CHR$17+CHR$3+CHR$230+CHR$8+CHR
$11+CHR$17+CHR$1+CHR$231
1610 E1$=CHR$32
1620 E2$=CHR$32+CHR$8+CHR$11+CHR$32
1630 E3$=E2$+CHR$8+CHR$11+CHR$32
1640 ENDPROC
```

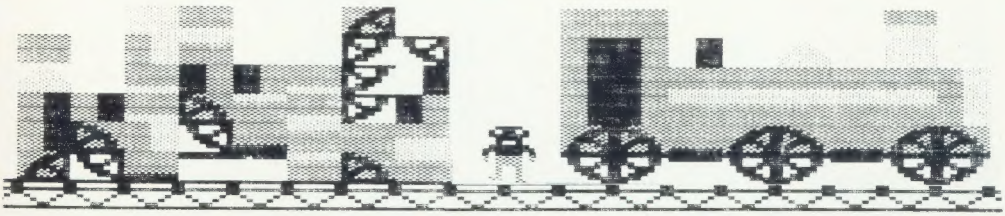


```
 1650 :
 1660 DEF PROCREBLK
 1670 BN%=Q% MOD16:C0%=(Q%DIV16)MOD4:C1%
=Q%DIV64:B$=CHR$17+CHR$(C0%)+CHR$17+CHR$
(128+C1%)+CHR$(235+BN%)+CHR$17+CHR$128
 1680 M3$=M2$+CHR$8+CHR$11+B$
 1690 ENDPROC
 1700 :
 1710 DEF PROCPUTBLK(LX%,LY%)
 1720  IF LY%<16 THEN LL%=LX%+(15-LY%)*4
0+1+Z% ELSE LL%=LX%+21+(30-LY%)*40+Z%
 1730 ?LL%=Q%
 1740 PRINTTAB(LX%,LY%);B$;
 1750 ENDPROC
 1760 :
 1770 DEF PROCUNBLK(LX%,LY%)
 1780 IF LY%<16 THEN LL%=LX%+(15-LY%)*40
+1+Z% ELSE LL%=LX%+21+(30-LY%)*40+Z%
 1790 Q%=?LL%:PROCREBLK:?LL%=0
 1800 SC%=SC%-10:PRINTTAB(LX%,LY%);E1$;
 1810 ENDPROC
 1820 :
 1830 DEF PROCMAN
 1840 IF IM%=0 THEN IM%=1 ELSE IM%=0
 1850 K%=X%+1+(Y%DIV16)*20+A%:M%=0:COLOU
P1
 1860 IF NOT(INKEY-104) THEN 1890
 1870  M%=1:H%=K%?-1:d%=?K%-H%:IF B%=4
AND X%<>0 AND H%<10 THEN B%=0:K%?-1=H%+1
:PROCPUTBLK(X%-1,Y%+d%):PRINTTAB(X%,Y%-2
);E1$;:SOUND0,-10,4,2:GOTO 1920
 1880  IF B%=0 AND H%>0 AND X%<>0 THEN B
%=4:K%?-1=H%-1:PROCUNBLK(X%-1,Y%+d%+1):S
OUND0,-10,4,2:GOTO 1920
 1890 IF NOT(INKEY-105) THEN 1920
 1900 M%=2:H%=K%?+1:d%=?K%-H%:IF B%=4 AN
D X%<>19 AND H%<10 THEN B%=0:K%?1=H%+1:P
ROCPUTBLK(X%+1,Y%+d%):PRINTTAB(X%,Y%-2);
E1$;:SOUND0,-10,4,2:GOTO 1920
 1910 IF B%=0 AND H%>0 AND X%<>19 THEN B
%=4:K%?+1=H%-1:PROCUNBLK(X%+1,Y%+d%+1):S
OUND0,-10,4,2:GOTO 1920
 1920 IF INKEY-67 THEN IX%=1:M%=2
 1930 IF INKEY-98 THEN IX%=-1:M%=1
 1940 IF B%=0 THEN M$=M2$:ES=E2$ ELSE M$
=M3$:E$=E3$
 1950 PROCMANCH:IF IX%=0 GOTO 2000
 1960 D%=?K%-?(A%+1+(X%+(Y%DIV16)*20+IX%
+40)MOD40):IF ABS(D%)>1 IX%=0:D%=0
 1970 PRINTTAB(X%,Y%);E$;
 1980 Y%=Y%+D%:D%=0:X%=X%+IX%:X%=(X%+20)
MOD20
 1990 IF (X%=0 AND IX%=1)OR(X%=19 AND IX
%=-1) THEN Y%=Y%+15*SGN(16-Y%)
 2000 IX%=0:PRINTTAB(X%,Y%);M$;
 2010 ENDPROC
 2020 :
 2030 DEF PROCINST
 2040 VDU19,1,1,0,0,0:VDU19,2,3,0,0,0:VD
U19,3,5,0,0,0
 2050 CLS:COLOUR3:PRINTTAB(2,2)" JASON
 MASON";
 2060 COLOUR1:PRINTTAB(1,6)"Collect from
  HERE";TAB(14,7);STRING$(6,CHR$228);
 2070 PRINTTAB(1,9)"and build     HERE";
TAB(14,10);STRING$(6,CHR$227);
 2080 COLOUR130:PRINTTAB(15,10);STRING$(
4,CHR$226);
 2090 COLOUR2:COLOUR128:PRINTTAB(6,12)"C
ONTROLS";
 2100 COLOUR1:PRINTTAB(0,15)"Z - LEFT  X
 - RIGHT";
 2110 COLOUR2:PRINTTAB(2,18)"PUT OR TAKE
 BLOCK";
 2120 COLOUR1:PRINTTAB(4,21)"> - From LE
FT";TAB(4,24)"? - From RIGHT";
 2130 COLOUR2:PRINTTAB(2,27)"D - Display
 Object";:COLOUR3:PRINTTAB(2,30)"RETURN
 to Continue";
 2140 REPEAT:UNTIL INKEY-74
 2150 CLS:PRINTTAB(0,1)"SELECT OBJECT (0
TO9)";
 2160 FOR I%=0TO9:RESTORE(I%+120.5)*20:R
EAD N$:IF HSC%(I%)>0 THEN COLOUR1:PRINT'
''" ",HSC%(I%):PRINTTAB(16,2)"TOP";TAB(1
5,3)"SCORE";
 2170 COLOUR2:PRINTTAB(2,4+I%*3);I%;SPC3
;N$;:NEXT:REPEAT:O%=INKEY(10):UNTIL O%>4
7 AND O%<58
 2180 RESTORE (O%+72.5)*20:READ N$
 2190 SC%=0:MEN%=3:ENVELOPE 1,1,1,0,-2,5
0,10,20,10,2,0,-1,120,110
 2200 ENDPROC
 2210 :
 2220 DEF PROCCHECK
 2230 IF INKEY-51 THEN SC%=SC%-100:PROCD
ISP:ENDPROC
 2240 IF OSC%<>SC% AND SC%>=0 THEN OSC%=
SC%:COLOUR128:COLOUR3:PRINTTAB(11,1);SC%
;SPC2; ELSE IF SC%<0 THEN FIN%=TRUE
```





```
 2250 EQ%=TRUE:FOR I%=X1% TO X2%:IF R%?(
RN%+I%)<>Z%?(RN%*4+I%+26) THEN EQ%=FALSE
 2260 NEXT:IF EQ%=TRUE THEN RN%=RN%+10 E
LSE RN%=0
 2270 IF RN%<V%*10 THEN ENDPROC ELSE COL
OUR3:PRINTTAB(4,18)"WELL DONE";:TIME=0:R
EPEAT UNTIL TIME>140:FIN%=TRUE:HSC%(O%-4
8)=SC%
 2280 PRINTTAB(1,18)"RETURN TO CONTINUE"
;:REPEAT:UNTIL INKEY-74:ENDPROC
 2290 :
 2300 DEF PROCDISP
 2310 OQ%=Q%:FOR I%=0 TO 99
 2320 Q%=R%?I%:PROCREBLK:PRINTTAB(5+I%MO
D10,30-I%DIV10);B$;
 2330 NEXT
 2340 TIME=0:REPEAT UNTIL TIME>160
 2350 FOR I%=0 TO 99
 2360 Q%=Z%?(40*(I%DIV10)+I%MOD10+26):PR
OCREBLK:PRINTTAB(5+I%MOD10,30-I%DIV10);B
$;
 2370 NEXT
 2380 Q%=OQ%:PROCREBLK
 2390 ENDPROC
 2400 :
 2410 DATA "EXPRESS",7,4,6,0,9,5,61,21,2
0,21,20,21,20,31,31,45,237,237,237,237,2
37,237,237,237,237,33,222,222,218,218,21
8,218,218,222,49,0
 2420 DATA 188,188,240,240,240,240,240,4
9,0,0,55,55,30,30,30,30,55,0,0,0
 2430 DATA "LINER",4,7,6,0,9,4,190,190,1
90,190,190,190,190,190,190,30,157,157,15
7,157,157,157,157,157,157,128,38,235,235
,235,235,235,235,33,0,0,0,46,96,39,96,39
,39,0,0,0
 2440 :
 2450 DATA "TAPESTRY",1,4,7,2,7,6,219,21
9,219,219,219,219,219,156,80,80,156,219,
219,80,177,176,80,219,219,80,224,225,80,
219,219,156,80,80,156,219,219,219,219,21
9,219,219
 2460 :
 2470 DATA "BOAT",7,4,6,1,9,4,238,238,23
8,238,238,238,238,238,46,240,240,199,240
,199,240,199,240,192,46,46,65,25,25,17,0
,0,0,0,23,23,23,23,0,0,0,0
 2480 :
 2490 DATA "BUNGALOW",1,7,3,2,8,7,24,24,
24,240,24,24,24,24,41,24,240,24,41,24,24
,41,24,240,24,41,24,24,24,24,24,24,24,24
,16,80,80,80,80,80,17,0,16,80,80,80,209,
0,0,0,0,0,0,240,0
 2500 :
 2510 DATA "CHURCH",1,7,3,0,9,9,188,188,
188,188,188,188,188,188,31,188,208,188,111,1
88,111,188,111,188,31,188,240,188,218,18
8,218,188,218,188,138,188,240,188,188,18
8,188,188,188,188,188,188,240,16,80,80,8
0,80,80,188,111,188,240
 2520 DATA 0,16,80,80,80,80,188,218,188,
240,0,0,43,0,0,0,188,188,188,240,0,0,0,0
,0,0,188,224,188,240,0,0,0,0,0,0,32,176,
32,176
 2530 DATA "TRAIN",7,4,2,1,8,6,21,20,29,
21,20,29,21,20,214,211,222,214,211,222,2
14,211,114,210,237,237,237,237,237,178,1
14,210,238,238,238,238,238,178,114,210,6
6,0,42,0,160,0,240,240,0,0,0,0,231,0
 2540 :
 2550 DATA "BUS",1,4,7,0,9,6,29,37,36,29
,29,29,37,36,29,29,222,80,80,80,80,80,80
,80,76,76,240,60,60,60,60,60,60,60,76,76
,80,80,80,80,80,80,80,80,80,80,60,60,60,
60,60,60,60,60,60,60,23,23,23,23,23,23,2
3,23,23,23
 2560 DATA 61,193,49,0,0,210,240,0,0,32,
193,49,0,0,0,210,57,0,0,0,49,0,0,0,0,23,
23,0,0,0
 2570 DATA "HOUSE",1,7,2,1,8,8,152,152,1
52,226,178,152,152,152,152,57,152,226,17
8,152,57,152,152,57,152,226,178,152,57,1
52,152,152,152,152,152,152,152,152
 2580 DATA 190,190,190,190,190,190,190,1
90,96,60,60,97,96,60,60,97,16,96,97,17,1
6,96,97,17,0,16,17,0,0,16,17,0
 2590  DATA "CASTLE",4,7,3,0,9,8,236,27,
27,27,236,236,236,236,236,236,236,27,27,
27,236,236,46,236,46,236,236,27,27,27,23
6,236,138,236,138,236,236,236,236,236,23
6,236,236,236,236,236
 2600 DATA 236,46,236,46,236,183,0,183,0
,183,236,138,236,138,236,0,0,0,0,0,236,2
36,236,236,236,0,0,0,0,0,183,0,183,0,183
,0,0,0,0,0
```



BEEBUG MAGAZINE is produced by BEEBUG Publications Ltd.

Editor: Mike Williams
Assistant Editor: Geoff Bains
Production Assistant: Yolanda Turuelo
Technical Assistant: Alan Webster
Secretary: Debbie Sinfield
Managing Editor: Lee Calcraft
Additional thanks are due to Sheridan Williams, Adrian Calcraft, John Yale and Tim Powys-Lybbe.



Editorial Address

**BEEBUG**
**PO BOX 50,**
**Holywell Hill,**
**St. Albans AL1 3YS**

CONTRIBUTING TO BEEBUG
PROGRAMS AND ARTICLES

We are always seeking good quality articles and programs for publication in BEEBUG. All contributions used are paid for at up to £40 per page, but please give us warning of anything substantial that you intend to write. A leaflet, 'Notes of Guidance for Contributors' is available on receipt of an A5 (or larger) SAE.

In the case of material longer than a page, we would prefer this to be submitted on cassette or disc in machine readable form using "Wordwise", "View", or other means, but please ensure an adequate written description of your contribution is also included. If you use cassette, please include a backup copy at 300 baud.

HINTS

There are prizes of £5 and £10 for the best hints each month, plus one of £15 for a hint or tip deemed to be exceptionally good.

Please send all editorial material to the editorial address above. If you require a reply it is essential to quote your membership number and enclose an SAE.

## SUBSCRIPTIONS

Send all applications for membership, subscription renewals, subscription queries and orders for back issues to the subscriptions address.

## MEMBERSHIP SUBSCRIPTION RATES

£ 6.90 6 months (5 issues) UK ONLY)
£12.90 UK - 1 year (10 issues)
£19.00 Europe, £22.00 Middle East
£24.00 Americas & Africa, £26.00 Elsewhere

BACK ISSUES
(Members only)

| Vol | Single issues | Volume sets (10 issues) |
|---|---|---|
| 1 | 90p | £8 |
| 2 | £1 | £9 |
| 3 | £1.20 | £11 |
| 4 | £1.20 | – |

Please add the cost of post and packing as shown:

| DESTINATION | First issue | Each subsequent issue |
|---|---|---|
| UK | 30p | 10p |
| Europe | 70p | 20p |
| Elsewhere | £1.50 | 50p |

All overseas items are sent airmail (please send a sterling cheque). We will accept official UK orders but please note that there will be a £1 handling charge for orders under £10 that require an invoice. Note that there is no VAT on magazines.

Back issues are for members only, so it is ESSENTIAL to quote your membership number with your order. Please note that the BEEBUG Reference Card and BEEBUG supplements are not supplied with back issues.

Subscriptions, Back Issues &
Software Address

**BEEBUG**
**PO BOX 109**
**St. Johns Road**
**High Wycombe HP10 8NP**

Hotline for queries and software orders

St. Albans (0727) 40303
Manned Mon-Fri 9am-4.30pm

24hr Answerphone Service for Access and Barclaycard orders, and subscriptions
Penn (049481) 6666

If you require members' discount on software it is essential to quote your membership number and claim the discount when ordering.

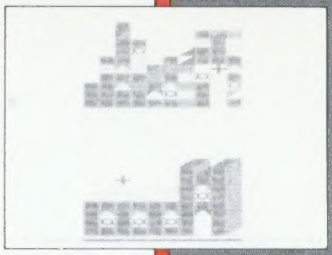
# Magazine Cassette/Disc

## CONTENTS OF THE MAGAZINE CASSETTE/DISC FOR APRIL 1986

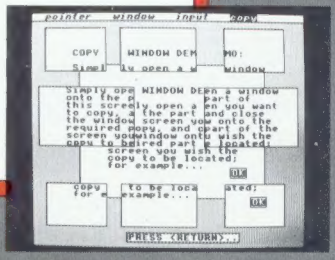
**INBETWEENING** – complete program and demonstration of this fascinating technique.
**DO-IT-YOURSELF BASIC** – add all the features that Basic lacks.
**DIRECT DISPLAY UTILITY** – list programs direct from tape or disc.
**FIRST COURSE** – useful example on hexadecimal conversion.
**BEEBUG WORKSHOP** – fast and accurate interrupt driven clock.
**GETTING A BETTER VIEW** – all the function key definitions ready to install.
**WINDOWS AND ICONS** – all the USI routines plus updated demo program.
**JASON MASON** – a game with a difference as you race to turn a pile of assorted building blocks into a recognisable object.

**ESTRA FEATURES THIS MONTH**
**MAGSCAN** – data for this issue of BEEBUG (Vol. 4 No. 10)

Inbetweening



Jason Mason




Windows and Icons

All this for £3.00 (cass) £4.75 (disc) + 50p p&p
Back issues (disc since Vol. 3 No. 1, cass since Vol. 1 No. 10) available at the same prices.

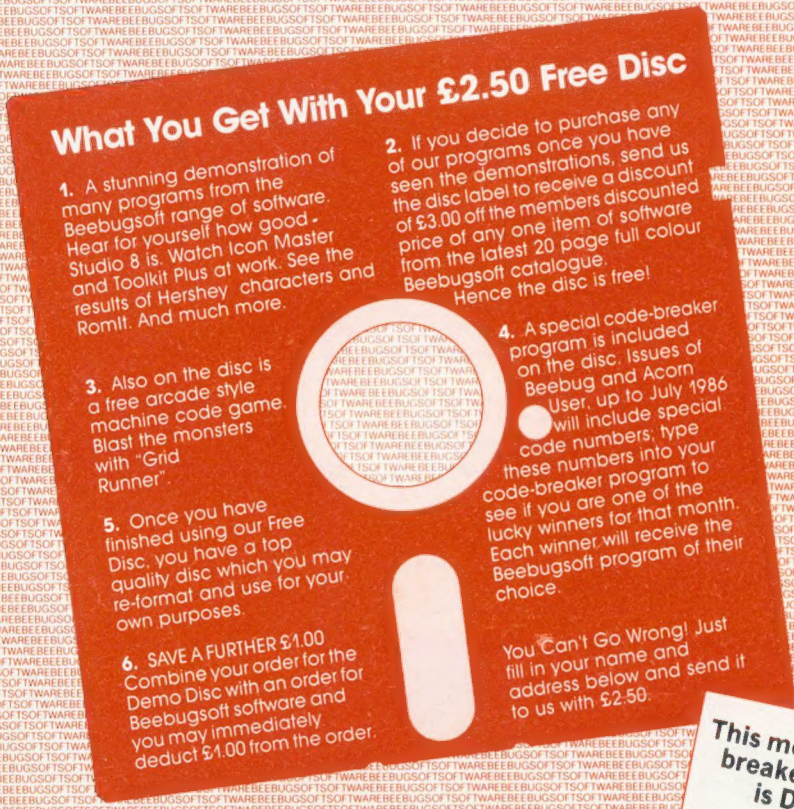
| Subscription rates | DISC UK | CASS UK | DISC O'seas | CASS O'seas |
|---|---|---|---|---|
| 6 months (5 issues) | £25.50 | £17 | £30 | £20 |
| 12 months (10 issues) | £50 | £33 | £56 | £39 |

Prices are inclusive of VAT and postage as applicable. Sterling only please.

Cassette subscriptions can be commuted to disc subscription on receipt of £1.70 per issue of the subscription left to run.

All subscription and individual orders to
**BEEBUG, PO BOX 109, St. Johns Road, High Wycombe HP10 8NP**

# Why Pay £2.50 for a FREE Demo Disc?




This offer is limited to one disc per household/institution

Please send me a "free" disc,
I enclose a cheque for £2.50/Please debit my Barclaycard/Access No.
OR I also enclose an order for software, please charge only £1.50

Name

Address

Specify 40 or 80 Track

Send to:
Beebugsoft,
PO Box 109,
High Wycombe,
Bucks., HP10 8NP